A Research UNIX Reader: Annotated Excerpts from the Programmer's Manual, 1971-1986

M. Douglas McIlroy

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

Selected pages from the nine research editions of the *UNIX® Programmer's Manual* illustrate the development of the system. Accompanying commentary recounts some of the needs, events, and individual contributions that shaped this evolution.

1. Introduction

Since it began at the Computing Science Research Center of AT&T Bell Laboratories in 1969, the UNIX system has exploded in coverage, in geographic distribution and, alas, in bulk. It has brought great honor to the primary contributors, Ken Thompson and Dennis Ritchie, and has reflected respect upon many others who have built on their foundation. The story of how the system came to be and how it grew and prospered has been told many times, often embroidered almost into myth. Still, aficionados seem never to tire of hearing about how things were in that special circle where the system first flourished.

This collection of excerpts from the nine editions of the research *UNIX Programmer's Manual* has been chosen to illustrate trends and some of the lively give-and-take—or at least the tangible results thereof—that shaped the system. It is a domestic study, aimed only at capturing the way things were in the system's original home. To look further afield would require a tome, not a report, and possibly a more dispassionate scholar, not an intimate participant. The raw readings are supplemented by my own recollections as corrected by the memories of colleagues.

The collection emphasizes development up to the Seventh Edition (v7*) in 1979, providing only occasional peeks ahead to v8 and v9. Although people elsewhere in Bell Labs and in Berkeley made significant contributions before v7, it was really with v7 that the system fledged and left the research nest. V7 was the first portable edition, the last common ancestor of a radiative explosion to countless varieties of hardware. Thus the history of v7 is part of the common heritage of all UNIX systems, while v8 and v9 will be more or less foreign to many readers. Moreover v7 happened long enough ago to be viewed with reasonable perspective.

The system was already well developed before v1 appeared in 1971. And not until v4 was the system first described in public [12], [13]. The technical side of the development from a paper exercise through a fully self-supporting model on a DEC PDP-7 to implementation on a series of PDP-11s has best been told by Ritchie in [11]. In his Turing Award lecture Ritchie reflected further upon the nature of the lab environment that fostered the development [10].

To keep the size of this report in bounds current versions of manual pages are rarely shown;

* For brevity I have adopted the designation v*n* for the *n*th Edition. This usage sprang from a colloquial tendency to refer to the Sixth and Seventh Editions as Versions 6 and 7, which inevitably led to the nickname v8 for the Eighth Edition. The rest followed.

interested readers will be familiar with and doubtless own the manual for v7 or one of its many descendants, such as System V or BSD. Many of the pages were chosen to show things happening, and some to show foibles. Enduring central features have been correspondingly slighted. By overlooking the news (or nonnews) about permanent things, I have unfortunately also overlooked the news (or nonnews) about just how well Thompson and Ritchie wrought. As many people have observed, the success of the UNIX system often owes as much to what it does *not* have as what it does. In the same way, lasting truths are likely not to be found in this story about its changes. Those I have tried to tell elsewhere [7].

In condensing the flow of events into comprehensible and compact history, I have largely overlooked, if indeed I could even have recognized, the myriad of borrowings of style and viewpoint and the continual interplay of criticism and code-dabbling that knit a cohesive *gestalt*. Without the vision of Ken Thompson, UNIX would not have come into existence; without the insight of Dennis Ritchie, it would not have evolved into a polished presence; without the imagination of Mike Lesk and popularizing touch of Brian Kernighan, it would not have acquired the extroverted personality that commands such widespread loyalty. But without any one of the people whose contributions are cited here, neither UNIX nor the work of others in the group would be the same.

1.1. The People

Up to v7 the *dramatis personae* were relatively few. Never officially recognized in any organization chart, the group coalesced voluntarily. Most of these original contributors are still with the Computing Science Research Center or its divestiture-induced clone at Bell Communications Research.

Ken Thompson began the construction from the ground up based on a file system model worked out with Ritchie and Rudd H. Canaday. He made processors for B, *bas*, and Fortran besides the operating system proper, and personally installed customized versions of UNIX for early clients as far away as Georgia. With Ritchie, Ken wrote the first shell and piles of utilities including *ed*, *roff*, *sort*, *grep*, *uniq*, *plot*, *sa*, and *dd*. He is also known for creative decompiling of mystery code. As if all this weren't enough, he has at the same time written circuit-optimizing tools, switching and network code, C compilers, and the basic software for several special-purpose machines, especially the chess champion, Belle.

Dennis M. Ritchie, best known as the father of C, joined Ken very early on. Dennis contributed basic notions such as *fork-exec* and set-userid programs. They jointly wrote the *fc* compiler for Fortran IV. The first debugger *db* and the definitive *ed* were Ritchie's, as was the radically new stream basis for IO in v8 and much networking software. With Steve Johnson he made UNIX portable, moving the system to an Interdata machine (v7). The names of Ritchie and Thompson may safely be assumed to be attached to almost everything not otherwise attributed.

Joe (Joseph F.) Ossanna, with the instincts of a motor pool sergeant, equipped our first lab and attracted the first outside users. Joe's *nroff* and *troff* indelibly shaped UNIX word processing and typesetting.

Bob (Robert) Morris stepped in wherever mathematics was involved, whether it was numerical analysis or number theory. Bob invented the distinctively original utilities *typo*, and *dc-bc* (with Lorinda Cherry), wrote most of the math library, and wrote *primes* and *factor* (with Thompson). His series of *crypt* programs fostered the Center's continuing interest in cryptography.

Doug (M. Douglas) McIlroy exercised the right of a department head to muscle in on the original two-user PDP-7 system. Later he contributed an eclectic bag of utilities: *tmg* for compiler writing, *speak* for reading text aloud, *diff*, and *join*. He also collected dictionaries and made tools to use them: *look* (v7, after a model by Ossanna), *dict* (v8), and *spell* (v7).

Lorinda L. Cherry collaborated with Morris on *dc-bc* and *typo*. Always fascinated by text processing, Lorinda initiated *eqn* and invented *parts*, an approximate parser that was exploited in the celebrated Writer's Workbench®, *wwb* (v8).

Steve (Stephen C.) Johnson's yacc reduced *Al (Alfred V.) Aho's* expertise in language theory to practice. Upon that base Steve built the portable C compiler that was used to port the system

itself and to evaluate candidate instruction sets for unbuilt machines. Johnson made the first *spell*, worked on computer algebra, and devised languages for VLSI layout.

Lee E. McMahon's linguistic insight fostered the characteristic text-processing—as distinct from text-formatting—capabilities of the system. He wrote *comm*, *qsort*, *sed*, the current *grep*, and the concordance-builders *index* for English and *cref* for C. After an early fling with *cu* and an influential laboratory switching system done with Condon, Morris, Thompson, *Chuck (Charles B.) Haley* and Cherry, Lee became the prime software architect for Sandy Fraser's Datakit® switch.

Brian W. Kernighan, expositor par excellence, coined the name UNIX, popularized the tools philosophy [6], wrote the best tutorials, and became a prolific inventor of specialized "little languages": *ratfor*, *eqn*, *awk* and *pic*. The Center's typesetting guru since the untimely death of Joe Ossanna, Brian produced the current "device-independent" version of *troff* and postprocessors for particular hardware.

Steve (Stephen R.) Bourne arrived at the time of v6, bringing Algol 68 with him. His definitive programs, the debugger *adb*, and the "Bourne shell," although written in C, looked like Algol 68: Steve wrote `DO-OD` and `BEGIN-END` instead of { and }. Bourne also contributed macro constructs to the UNIX Circuit Design System (see 4.3).

Mike (Michael E.) Lesk, with a prescient market instinct, made text formatting accessible to the masses with the generic macros *−ms*, which were to *troff* what a compiler is to assembly language. He rounded out *−ms* with the preprocessors *tbl* for typesetting tables and *refer* for bibliographies. He also made the *lex* generator of lexical analyzers. Eager to distribute his software quickly and painlessly, Mike invented *uucp*, thereby begetting a whole global network. Over the years, often helped by *Ruby Jane Elliott*, he initiated fascinating on-line audio, textual, and graphical access to phone books, news wire (*apnews*, v8), *weather* (v8), and UNIX instruction (*learn*, with Kernighan, v7).

Stu (Stuart I.) Feldman implemented, with *Andy (Andrew D.) Hall, the efl* preprocessor to sugar Fortran with PL/I-ish syntax. He wrote the *f77* Fortran compiler single-handedly and invented the famous *make*. A man of taste and culture, Stu exhibited both in his underground classic on UNIX style [3].

Peter J. Weinberger has moved effortlessly among number theory, databases, languages, and networking. Weinberger boasts the middle initial of *awk*, the IO library for *f77*, and a famous visage (see FACED, page 14). In v8 and v9 appeared an unbounded precision arithmetic package *mp*, a fast factoring program *qfactor*, a B-tree library *cbt*, and a new code generator for C, all Peter's work. Above all, his network file system bound a stable of machines together into a logically homogeneous system (v8).

Sandy (A. G.) Fraser devised the Spider local-area ring (v6) and the Datakit switch (v7) that have served in the lab for over a decade. Special services on Spider included a central network file store, *nfs*, and a communication package, *ufs*. Datakit, a "central office" for data communication, gave added impetus to research in distributed computing. Fraser undertook the Unix Circuit Design System (see CDL in section 4.3) to support his hardware projects.

Joe (Joseph H.) Condon, physicist and circuit designer extraordinaire, although not usually listed as an "author" of UNIX, is an indispensable presence among the group. He wrote much of the Unix Circuit Design System, including sophisticated wire-routing algorithms. He designed superfast specialized machines, including the chess machine Belle (with Thompson), domesticated real telephone switches to our laboratory environment for the experiments of McMahon and others, and made unusual connections to the telephone system that, among other things, let our lab computer detect and announce incoming voice calls.

Al (Alfred V.) Aho's unstintingly given insight into language theory and algorithms shows up in programs by many people, such as *yacc*, *lex*, *cc*, the Writer's Workbench, and the screen editor *sam*, as well as in his own *awk*, *egrep*, and *fgrep*.

Greg (Gregory L.) Chesson pursued all aspects of computer-to-computer communication: multiplexing with *mpx* (v7), flow-controlled channels with *con* (v7) and *dcon*, and protocols, including one used by *uucp*. The first kernel- and user-level software for Datakit was his.

Many other people contributed. By the time of v4 the role of provider to the by then sizable clientele within Bell Labs had been assumed by *Berk (Berkley A.) Tague* and his UNIX Support Group, thereby guaranteeing the future of the system. Shortly thereafter the Programmer's Workbench project undertook to adapt the system to support large software efforts. Their concern with administrative tools led to somewhat differently flavored utilities such as *find* (v5), *cpio* (v7), and *sccs*. *Joe (Joseph F.) Maranzano* of the USG and *Dick (Richard C.) Haight* of PWB became *de facto* adjunct members of the research group. Haight contributed *find*, *cpio*, and *expr*, all in v7. *Ted (Theodore A.) Dolotta* of PWB did much to refine the manual.

Some USG and Computer Center people eventually joined the Computing Science Research Center to bring order to our zoo of equipment: *Andy (Andrew R.) Koenig* conceived and built *asd*, the automatic software distribution system that keeps v9 current across about 50 different computers, and *snocone*, a preprocessor to sugar the syntax of Snobol into a structured language. *Fred (Frederick T.) Grampp's* unrivaled practical experience in computer security shows up in subtle countermeasures now routinely used in our systems. A thorough security audit program of his ultimately became the administators' tool *quest*. *Ed (Edward J.) Sitar*, with devotion worthy of Ossanna, saw to it that the hardware actually worked, keeping twenty machines housed and powered, and suppliers on their mettle.

Tom (Thomas B.) London and *John F. Reiser* ported v7 to the VAX and introduced paging. Their V32 system, as filtered through Berkeley, became the progenitor of almost all research UNIX systems. Reiser later contributed a peerless compile-and-execute *bitblt* for the Teletype 5620.

People who joined the research center after v7 led development in new directions. *Bart N. Locanthi* designed the bitmapped terminal known variously as "jerq," "Blit," and Teletype 5620. He programmed it, too: from graphic primitives, such as the crucial *bitblt*, up to a multiterminal maze war game. *Rob Pike* supplied a multiprogramming system, host-terminal communications, mouse control, and support for overlapping virtual terminals (*mpx*, later *mux*). Pike's system fostered fascinating programs by many people, of whom I shall mention only a few (v8). Pike himself built visual editors, culminating with *sam* (v9), which went well beyond the command capability of *vi*—seemingly with no more mechanism than the venerable *ed*. (Rob's simplifying touch is also visible in the shell, in *p* for paginating, and in his recasting of Chesson's remote execution software, all in v8.)

The Blit terminal attracted *Mark S. Manasse* who (with Pike) invented *lens*, an algorithmically ingenious bitmap magnifier, and *tek4014*, a disarmingly faithful simulator of Tektronix terminals. *Luca Cardelli* contributed an *icon* builder, annoying *crabs* that devour screen images, and the catchy *vismon* that posts icons of the senders of incoming mail. *Tom (Thomas A.) Cargill* did a monumental multiview debugger *pi*, thoroughly exercising the object-oriented capabilities of *Bjarne Stroustrup's* C++. *Tom (Thomas J.) Killian* made a font editor *jf* and programs to save and print bitmap images (*blitblt*, *thinkblt*) that led in turn to *can*, a comprehensive suite of laser-printer software built from the ROM up. For the UNIX system itself, Killian invented the important `/proc` file system that contains core images of all running processes.

Dave (David L.) Presotto tamed networks. His *upas* brought some order to a Babel of mail addresses and his *ipc* primitives provided a common basis for communication and remote file access via Internet, Ethernet, and Datakit. *Bill (William T.) Marshall*, who shares principal Datakit software responsibility with McMahon, wrote basic protocol code. These protocols merit unusual confidence: their correctness was mechanically verified by *Gerard Holzmann*.

Andrew G. Hume wrote *proof* to put *troff* on your screen (v8, begun by Locanthi), parts of the UNIX Circuit Design System (v9), *mk* to supplant *make* (v9), and a remote *backup* service (v9). *Norman Wilson* shares with Ritchie the honors of reigning guru. He has been instrumental in porting v8 to a Cray and in rationalizing system configuration procedures. Waging a personal battle against entropy, Wilson rectified countless infelicities, glitches, and blunders in the software and the manual, always making things shorter and simpler as he did so.

1.2. The Manual

During the system's first two years one literally had to work beside the originators to learn it. But the stock of necessary facts was growing and the gurus' capacity to handle apprentices was limited. As they wished to spread the good news about their marvelous system, a manual became a necessity. Ritchie one day set forth the first "man page" in a format that has stood the test of time. The terse, yet informal, prose style and alphabetic ordering encouraged accurate on-line documentation: it was not a big deal to decide how and where to describe changes as they happened. The format was popular with initiates who needed to look up facts, albeit sometimes frustrating for beginners who didn't know what facts to look for.

The absence of any "logical" grouping of facilities was a deliberate result of discussion. (As encyclopedists have always known, the relationships among knowledge are too various to force into rational linear order.) Retrievability and honesty were the prime concerns. The refreshing BUGS notes served as a constant reminder of areas for improvement.

The first manual was duplicated for a very small coterie. In order to channel queries directly to the horses' mouths, authorship was attributed to individuals. Later, as authorship diffused, on the principle of "You touched it last; it's yours," authorship was attributed only in the segregated chapter of unofficial "user maintained programs." Beginning with v7, this back-of-the-bus chapter was reserved for games.

As the system was elaborated, peer pressure in the research group caused rough places to be smoothed, vague ideas to be sharpened, and feeble programs to be extinguished. Most details of the constant questioning and experimentation during the early period of rapid change are long forgotten, as are hundreds of transitory states that were recorded in the on-line manual. From time to time, however, a snapshot was taken in the form of a new printed edition. Quite contrary to commercial practice, where a release is supposed to mark a stable, shaken-down state of affairs, the very act of preparing a new edition often caused a flurry of improvements simply to forestall embarrassing admissions of imperfection.

1.3. The Events

Among the more memorable minirevolutions that the system experienced were

- The appearance of pipes elevated standard-in-standard-out design to the status of a "philosophy" (v3). Old software was gradually brought into line (see SORT on page 7).
- *Grep* (Thompson, v4) ingrained the tools outlook irrevocably. Already visible in utilities such as *wc* (Ossanna, v1), *cat*, and *uniq* (v3), the stream-transformation model was deliberately followed in the design of later programs such as *tr* (McIlroy, v4), *m4* (Kernighan and Ritchie, v7), *sed* (McMahon, v7), and a flurry of language preprocessors.
- Conversion to C (v4) made basic abstractions clearer. Assembly language and magic constants gradually declined from the status of the "real truth" (v4) to utterly forgotten (v8).
- The novel style of *eqn* influenced the design—even the conception—of a still growing generation of special purpose languages (v5).
- The move away from PDP-11s caused a further push for portable abstractions. The main visible symptom was a proliferation of include files (v7).
- The Bourne shell almost overnight drove out the simple old shell. A PWB shell had made shell programming useful; the Bourne shell made it an essential part of UNIX programming (v7).
- Mike Lesk's *uucp* gave operational meaning to the phrase "UNIX community" (v7). News now travels electronically among users all over the world; and technical collaborations proceed between distant locations almost as easily as within one building.
- Direct network connections among our computers began with Sandy Fraser's Spider network and became widespread with Datakit (Chesson and Ritchie, v7). Datakit and Ritchie's streams (v8) made possible Peter Weinberger's network file system, Andy Koenig's automatic software distribution, and Dave Presotto's connections to diverse networks. As a result "the"

research machine is no longer identifiable; users can—and do—work on one or more of two dozen computers simultaneously.

- Bitmapped terminals operating under Rob Pike's "jerq" software caused a quantum jump in personal multiprogramming and inspired intriguing new programming styles (v8).

The early editions came in quick succession. Later the interval between editions increased for several reasons. First, most of the system was mature and stable. Second, from the standpoint of the manual, much research was subliminal. For example, the biggest system changes from v6 to v7 to v8—portability and streams—barely affected the manual. Third, the need for timely printing diminished as other organizations became responsible for distribution. And fourth, as the system grew to encompass facilities beyond any individual's ken,* the task of organizing an ever-growing manual for printing became increasingly daunting.

The UNIX lab has always been an exciting place to work. As I recorded this summary, I recalled vivid individual moments when new ideas or startling combinations of old ideas flashed through the lab, when programming met theory and *vice versa*, and when advances on many simultaneous fronts built upon and reinforced one another. I was forcibly reminded over just how wide a spectrum of activities and interests each member of the group has ranged and how freely and without fanfare collegial help has flowed among them. The primary dividends to the participants have been the fun of doing, the joy of accomplishment and the satisfaction of seeing one's handiwork used. Intellectual proprietorship and physical ownership count for little; there's more than enough of both for everyone.

2. Primary Commands

CAT (v1 page 16†)

Cat is probably the oldest and best-known of all distinctively UNIX utilities. The current (v9) description scarcely differs from that in v1. On the PDP-7 there had been a program *pr* for copying a single file to the terminal. Having been subsumed by *cat*, *pr* was retired and its name was recycled.

Since *cat* was the prototypical filter, people were tempted to pile on options for other functions. Thus *cat* was pressed into service to block output into 512-byte chunks. That in turn led to `cat -u` (v7) to turn the feature off. This dismal admission that byte streams might not always be pure had been overcome by the time of v8. In other circles the chastity of *cat* was violated more severely; see Pike's essay on `cat -v`[9].

CP (v1 page 17, v5 page 18)

The war-horse utility *cp* and its close relative *mv* originally worked on lists of pairs. Such lists, however, could not be generated by the shell's * convention. All too often mistyped lists clobbered precious files. Consequently both utilities were promptly cut back to handle just one from-to pair (v2). At the same time *mv* was generalized to move files to a named directory. Strangely *cp* picked up only a BUG note suggesting the feature. By the time of v3 both had converged to their present forms, although an unexplained option `-t` intruded briefly in v5. What seems natural in hindsight was not clear cut at the time: the final conventions arose only after long discussions about how properly to handle file permissions and multiple files. In fact the discussion is not yet closed. Whether and how to recurse on directories is still debated: v7, v8, and v9 each offered a different way to do it.

---

* Ken's ken was probably the last to saturate. At the time of v5, shell accounting once revealed that Thompson had used 102 distinctly named programs in the course of a week. Nobody else came close.

† The *cat* page from v1 is reproduced on page 16 of this report.

SORT (v1 page 19, v5 page 20)

This mainstay utility began as a "user-supported progam" in chapter 6 of v1 (Thompson). Upon query to the author it turned out that the "wide options" announced there involved altering the source. The first official options appeared in v4. Expanding ever since, the load of options reached 17 in v9 (McIlroy and *John P. Linderman* of the Computer Technology Research Lab). The program originally sorted in core; pressure of real use soon forced it to spill to disk.

The first design, typical of pre-pipe days, had an argument to name the output (see GREP below):

```
sort input output
```

During the pipeline revolution Thompson modified *sort* to be usable as a filter (v4). Unlike most utilities, though, *sort* did retain an output-naming convention because it was so often used to sort a file in place, which couldn't be done with >.

When he extended *sort* to handle multiple files, Thompson invented a special name "-" for the standard input (v5). The convention caught on and soon infected many other commands. As a property of particular commands and not of the system as a whole, "-" itched naggingly. The itch went unscratched until Ritchie, at Pike's suggestion, installed `fd` special files synonymous with already open file descriptors (v8). The stubborn disease remains, however.

A bug note in *join*(1) declares, "The [field-specification] conventions of *join, sort, comm, uniq, look* and *awk*(1) are wildly incongruous." Although these programs are often used together, they remain, like American weights and measures, sturdily eccentric.

MAIL (v1 page 21, v7 page 22)

Electronic mail was there from the start. Never satisfied with its exact behavior, everybody touched it at one time or another: to assure the safety of simultaneous access, to improve privacy, to survive crashes, to exploit *uucp,* to screen out foreign freeloaders, or whatever. Not until v7 did the interface change (Thompson). Later, as mail became global in its reach, Dave Presotto took charge and brought order to communications with a grab-bag of external networks (v8).

Despite the turbulent evolution of *mail,* to this day a simple postmark is all that it adds to what you write. Old UNIX hands groan at the monstrous headers that come from latter-day mailers and at the fatness of their manuals.

ECHO (v2 page 23)

Echo, seemingly the simplest of utilities, originated with Multics, where it was used to test the sanity of the shell. The present version arose as a finger exercise in C programming (McIlroy, v2). Then it turned out to be useful, a mainstay of shell scripts.

For a while *echo* was complemented by *prompt* (never documented), which did the same thing without a newline. Eventually *prompt* was displaced by `echo -n` (Ritchie, v7). Meanwhile a minor echo-amplification industry arose in some quarters. Imported versions of *echo* with elaborate syntax came and went in a midnight vendetta; for a time it was the least stable of all commands. Ritchie calmed the altercation with a Solomonic but un-UNIX-like option to switch between two competing syntaxes (v8). Not surprisingly, the more elaborate choice has never been needed in any shell script in `/bin` or `/usr/bin`. The whole episode inspired McIlroy's parable, "The Unix and the Echo," quoted in [5], page 79.

GREP (v4 page 24)

Grep, generally cited as *the* prototypical software tool, was born when Ken Thompson was asked how to search for patterns in a file that was too big for the editor. Thompson promptly liberated his regular expression recognizer and christened it after the *ed* command `g/re/p`. It was an instant hit and soon entered our language as a verb.

The success of *grep* suggested other utilities. One of these, *gres* for global substitution, evolved into the stream editor *sed* (McMahon, v7). *Grep* also inspired Al Aho to apply his

encyclopedic knowledge of language theory to make the very general *egrep* and the highly specialized *fgrep* (v7). Over the years Aho honed *egrep* into an awesomely performing program, simple on the outside but the highest of tech on the inside.

The v4 manual page for *grep* is typical of earlier editions. Options were described in running text along with the basic usage; the more readable convention of always displaying options separately, no matter how few a command might have, gained ground only slowly. In typical early style the synopsis permits an output argument as well as an input argument—poor human engineering, because of the disastrous consequences of invoking `command input output` as if were `command input1 input2`. This dangerous syntax was expunged from most commands by v7, but at least one fossilized instance survives in BSD 4.3.

3. Programming

3.1. The Shell

SH (v1 pages 25-27, v3 pages 28-33, v4 pages 34-36)

The name and the general outline of the "shell" originated in Multics, but for UNIX the shell had to be pared back to basics to fit in an 8K user space. There wasn't even enough room to do * name expansion; that task was handed off to another program *glob*, signifying "global" (Ritchie, v1). (*Glob*, written in B, was the first piece of mainline UNIX software to be done in a higher level language.)

The shell read commands from the same standard input as did programs that it invoked. Thus commands and data were interleaved in command files, or "runcoms,"* now usually called shell scripts. There was no way to mark the end of embedded data files; programs that buffered their input would consume input not intended for them. Consequently programs that were likely to read from shell scripts, especially *sh* and *ed*, were made to read their input one character at a time. It was impossible to pipe into a shell script because the standard input was already dedicated to the script. For the same reason a program in a shell script could not take input from a terminal except when given the terminal's real name. None of these problems was addressed until the Bourne shell solved them all at once (v7).

A shell notation for composing programs into pipelines accompanied McIlroy's proposal for pipes. This new idea, with its curious syntax, took almost a page to describe (v3). The syntax was reformed when, to avoid the embarrassment of describing it in a big public talk, Thompson proposed the appealing infix ¦. Thus naturalized, pipelines became describable in just four sentences in v4.

The UNIX shell gave up the Multics idea of a search path and looked for program names that weren't file names in just one place, `/bin`. Then in v3 `/bin` overflowed the small (256K), fast fixed-head drive. Thus was `/usr/bin` born, and the idea of a search path reinstated.

GOTO, : (v2 pages 37-38)

Goto manipulated the standard input to give the illusion of a programmable shell (Thompson, v2). The associated *:* command, which thanks to ASCII collating sequence boasted the first page in the manual, was a big nop.

Other flow-of-control programs were *if*, to execute a command conditionally (Ritchie and Thompson, v2), and *exit* (Thompson v2). With the luxury of bigger computers, Bourne made a genuinely programmable shell and abolished these clever but clumsy tricks (v7). Nevertheless *:* survives as a built-in shell command and *test* has inherited the boolean capabilities of *if*.

---

* *Runcom*, a program that could run a short script of commands in the background, was the closest thing MIT's CTSS had to a callable shell. A vestige of the name survives in the boot script, `/etc/rc`.

3.2. System Calls

STAT (v1 page 39, v4 page 40, v7 pages 41-42)

Stat is one of very few original system calls to have changed at all, mostly because through it user code glimpses system tables. In v4 group permissions appeared and file sizes went from 16 to 24 bits. The size change was tough; it meant rebuilding every existing file system, and the longer addresses compelled some trickery to avoid sacrificing space or performance on smaller files.

The snapshots of *stat* also show how the style of description evolved from assembly language (v1) through C (v4) to a more abstract and portable form with declarations hidden in include files (v7). The traditional sense of what constituted the real system interface eroded very slowly. Even though almost all programs were written in C, assembly language held pride of place in the manual through v6. Banished to a footnote in v7, assembly language did not disappear completely until v8, twelve years after the birth of C.

PIPE (v3 page 43)

The basic redirectability of input-output made it easy to put pipes in when Doug McIlroy finally persuaded Ken Thompson to do it. In one feverish night Ken wrote and installed the *pipe* system call, added pipes to the shell, and modified several utilities, such as *pr* and *ov* (see 5.1 below), to be usable as filters. The next day saw an unforgettable orgy of one-liners as everybody joined in the excitement of plumbing. Pipes ultimately affected our outlook on program design far more profoundly than had the original idea of redirectable standard input and output.

All programs placed diagnostics on the standard output. This had always caused trouble when the output was redirected into a file, but became intolerable when the output was sent to an unsuspecting process. Nevertheless, unwilling to violate the simplicity of the standard-input-standard-output model, people tolerated this state of affairs through v6. Shortly thereafter Dennis Ritchie cut the Gordian knot by introducing the standard error file. That was not quite enough. With pipelines diagnostics could come from any of several programs running simultaneously. Diagnostics needed to identify themselves. Thus began a never quite finished pacification campaign: a few recalcitrant diagnostics still remain anonymous or appear on the standard output.

The first implementation of pipes used a single file descriptor for both reading and writing (v3). The modern scheme, with two file descriptors, came in the next edition.

INTR (v1 page 44), SIGNAL (v4 page 45)

In v1 there was a separate system call to catch each of interrupt, quit, and two kinds of machine traps. Floating point hardware (v3) brought another and no end was in sight. To stop the proliferation, all traps were subsumed under a single system call, *signal* (v4). The various traps were given numbers, by which they were known until symbolic names were assigned in v7. We have not been fully weaned yet: the numbers are still needed in the shell *trap* command.

A simple unconditional *kill* was available to terminate rogue programs in the background (v2). In v5 *kill* was generalized to send arbitrary signals. Never, however, was the basically unstructured *signal-kill* mechanism regarded as a significant means of interprocess communication. The research systems therefore declined to adopt the more reliable, but also more complex, Berkeley signals.

In general, research UNIX systems, out of a belief that asynchrony is nasty, have provided less overt support for it than have some of their relatives. Thus when multiprocess coordination is unavoidable, as for receiving mail, curious synchronizing tricks with dummy files have been resorted to. The short-lived multiplexor (Chesson, v7), then Ritchie's `/dev/pt` named pipes and *select* (adapted from Berkeley) lent some support for these special needs (v8); Presotto's *ipc* code goes still further (v9).

The research systems remain standoffish about unbridled parallelism; the recent facilities for asynchrony are used only by cognoscenti. Nevertheless users of UNIX systems are probably more at home with parallel computing—as structured by pipes—than is almost any other user community.

STTY (v2 pages 46-47), IOCTL (v7 page 48)

Ioctl is a closet full of skeletons. The *ioctl* story began with *stty* (v2), the primary use of which—setting modes upon logging in—was unexceptionable. Trouble set in when other programs began to use it. These programs would work for their owners on their owners' terminals, but could fail frustratingly in other settings. Thus was the slippery slope to *curses* first glimpsed.

Stty accumulated features gradually, and often incompatibly. Eventually it was rechristened *ioctl* to keep abreast of plans for the corporate standard release 3.0 (v7). This faceless name, with no intrinsic meaning, quickly acquired more than enough. It was somehow exempt from the ethos of simplicity that kept the lid on new system calls. All kinds of functions were piled onto *ioctl*. The interface varied bewilderingly from function to function and from system to system. Documented willy-nilly throughout chapter 4 and sometimes only in source code, its true dimensions can never be appreciated.

3.3. Standard IO

PRINTF (v4 page 49)

Output formatting was originally left up to programmers (or to Fortan). Although v1 included conversion routines like *atof*, the *printf* routine that Ritchie had long since devised for BCPL did not arrive until the C change (v4). Formatted input was even slower in coming: Mike Lesk's portable IO library that included *scanf*, as well as *gets* and *ungetc*, did not become official until v7.

PUTC (v1 page 50), STDIO (v7 page 51)

Buffered IO was, and still is, a necessary evil. A rudimentary buffering package with *getc*() and *putc*() in v1 required the user to supply buffers and manage file descriptors. This scheme persisted until Ritchie's *stdio* reconciled the buffering package with Lesk's portable IO, hid the dependence on file descriptors, and eliminated per-character function calls. In one clean sweep *stdio* made C programs easily portable. In the ANSI draft standard for C *stdio* enjoys equal status with the language proper.

3.4. Languages

Almost everybody in the group has done languages. Thompson and Ritchie built assemblers from scratch. On the tiny PDP-7 the assembler was supplemented by *tmg*, Doug McIlroy's version of Bob McClure's compiler-compiler. Using it Thompson wrote B, the evolutionary link between Martin Richards's (Cambridge University) BCPL and C. The PDP-11 assembler, a desk calculator *dc*, and B itself were written in B to bootstrap the system to the PDP-11. Because it could run before the disk had arrived, *dc*—not the assembler—became the first language to run on our PDP-11. Soon revised to handle arbitrary-precision numbers (v1), *dc* was taken over by Bob Morris and Lorinda Cherry. It now ranks as the senior language on UNIX systems.

CC (v2 page 52)

As a testbed for floating-point routines in v1 Thompson wrote *bas*, a Basic-like interpreter. It survived as long as we used PDP-11s. V2 saw a burst of languages: a new *tmg*, a B that worked in both core-resident and software-paged versions, the completion of Fortran IV (Thompson and Ritchie), and Ritchie's first C, conceived as B with data types. In that furiously productive year Thompson and Ritchie together wrote and debugged about 100,000 lines of production code.

Conversion to C made UNIX, already elegant and capable, into a system also intelligible, pliable, and ultimately portable. It elicited a flood of utilities and made it easier to refine the kernel. As the compiler evolved, the system benefited too: better object code meant speedups and space savings across the board. More than once an overgrown kernel was squeezed back into place by attending to the compiler.

Portable utilities written in C spread easily to other computing environments at Bell Labs. Gradually users became able to deal with systems from several manufacturers on the same terms: programs like *ls*, *cp*, and above all *sh* worked similarly everywhere. With the human, if not the machine interfaces already established, the ultimate transition to *UX* systems proper on mainframes was remarkably gentle.

YACC (v3 page 53)

With *yacc* Steve Johnson reduced to practice Al Aho's expertise in language theory (v3). *Yacc*, abetted by Mike Lesk's *lex* (v7), stimulated a language industry. Among the better known *yacc*-based processors are

eqn for typesetting equations (Kernighan and Cherry, v5)

ratfor, which provided C-like syntax for Fortran (Kernighan, v6)

bc for arbitrary-precision computation (Morris and Cherry, v6)

m4, a general macroprocessor (Kernighan, v7)

apl, Iverson's language (Thompson, v7)

struct, convert Fortran to Ratfor (*Brenda S. Baker*, v7)

f77, a Fortran 77 compiler (Feldman and Weinberger, v7)

awk, a pattern-directed file-processing language (Aho, Weinberger and Kernighan, v7)

pcc, a portable C compiler (Johnson, v7)

pic for typesetting line drawings (Kernighan, v8)

ideal, a constraint-based language for typesetting line drawings (*Chris (Christopher J.) Van Wyk*, v8),

C++, an "object-oriented" extension of C (Bjarne Stroustrup, v8)

hoc, a C-like "desk-calculator" language (Kernighan and Pike, v8)

grap for typesetting graphs (Kernighan and *Jon L. Bentley*, v9)

Several of these languages are *tours de force*: *eqn* for its insightful syntax, *bc-dc* for its unique variable-precision math library, *struct* for finding both structure and theorems in undisciplined code, *ideal* for enlisting symbolic computation in the service of drafting. *Yacc*, by eliminating much drudgery of compiler-writing, made possible the extensive experimentation that underlies these novel languages. It is no exaggeration to say that without *yacc* some would never have been undertaken, and many would not have evolved into more than mere demonstrations.

4. Applications

4.1. Text Processing

Among the early justifications for the research activity that produced the UNIX system was the potential for text-processing. Thus the first significant application program was *roff*, which printed the manuals for v1, v2, and v3 (Thompson and Ritchie) and attracted the first outside client, the patent department at Bell Labs.

Thereafter Joe Ossanna became the driving force behind UNIX text formatting software. His macro-based, trap-driven *nroff* appeared in v2. When Graphics Systems, Inc., announced an inexpensive typesetter with ASCII paper tape input, Ossanna sprang for one, replaced the tape reader with a wire to the computer, and modified *nroff* for multiple fonts and proportional spacing. *Voilà, troff*. It blew the manufacturer's mind, and touched off a flurry of homemade documents in flamboyant layouts—good enough, however, to fool referees into suspecting that the manuscripts had been published before.

Ossanna's ultimate intent, that macros should foster higher-level typsetting languages, was finally realized with the invention of the `-ms` macros (Lesk, v7, but dating from 1976).

FORM (v1 page 54), TYPO (v3 page 55)

"Text processing" means considerably more than mere text formatting. The elegant *form* letter generator was written by Thompson from Morris's original on CTSS (v1). Augmented by a special editor *fed* (Cherry, v2) *form* provided a genuine personal database. McMahon, Morris, and Cherry together collected a substantial corpus of text and studied it statistically [8]. Some of their tools, particularly *uniq* (Thompson, v3) and *comm* (McMahon, v4) became staples. Out of that work came the remarkable *typo*, which spotted typing errors by statistical inference (Morris and Cherry, v3). Eventually Steve Johnson's *spell*, a virtuoso demonstration of tools in use (see page 126 of [6]), shouldered *typo* aside (v5) and spurred McIlroy to engineer a sophisticated version (v7).

Form was really a macroprocessor with persistent memory. The long tradition of macros at Bell Labs assured that others would appear: *m6* by McIlroy, Morris and Andrew D. Hall, then *m4* (Kernighan and Ritchie). And macros of course were central to the text formatters, *roff*, *nroff*, and *troff*.

OV (v3 page 56)

Ov did multicolumn formatting cheaply and elegantly (Ossanna, v3). One printed a narrow column with every other page offset one-half a page width and then ran the output through *ov*, which or-ed pairs of pages together. The following 4-column wonder was shown off on the first day of pipes:

```
roff file >ov>ov>
```

which means in modern language, `roff file | ov | ov`. *Ov* was ultimately killed off by `pr -n` and features built into *nroff* (v5).

WWB (v8 page 57)

Cherry's work on approximate parsing and Aho's on fast pattern search turned out to be just the right foundation for an English style-appraiser suggested by Prof. William Vesterman of Rutgers. That in turn was elaborated into Writer's Workbench by *Nina (Antonina H.) MacDonald* and others in the Human Performance Engineering Department (v8). WWB attracted unusual attention from the popular press, including the New York Times and the Today show.

4.2. Navigation

SKY (v4 page 58)

By some quirk of providence, many members of the group have been fascinated by navigation, geodesy, and astronomy. The first celestial program was Ossanna's satellite predictor, *azel*, which had guided the ground station for Telstar (v2). Morris's, then Thompson's *sky* programs predicted everything else, giving daily voice and mail announcements like, "Eclipse of the moon at 8:42 PM." McIlroy's *map*, intended to display the earth on dozens of projections (v7), was used by McMahon to portray the heavens. Upon databases of maps, stars, cities, airports, and weather were built route planners for car (*blitmap*, Lesk and Elliott, v8) and plane (Thompson) and an astronomer's informant (*scat*, Pike, v9).

WWV (v8 page 59)

Already in v1 were routines, *cal* (Thompson) and *ctime* (Ritchie), that knew calendrical facts well beyond any immediate system need. In the 1980s, as the lab's work spread across a network of machines, the many aberrent clocks became intolerable. Andy Koenig made the most of a cheap and jittery receiver for the National Bureau of Standards broadcast time standard *wwv*, arranging for individual clocks to adjust to the standard smoothly, without backdating. Later Condon installed and Thompson programmed a robust filter for a raw time receiver on an almost uninterruptable computer called "the rock" (v9).

4.3. Design Automation

CDL (v7 pages 60-63)

Although most users do not encounter the UNIX Circuit Design System, it has long stood as an important application in the lab. Originated by Sandy Fraser and extended by Steve Bourne, Joe Condon, and Andrew Hume, UCDS handles circuits expressed in a common design language, *cdl*. It includes programs to create descriptions using interactive graphics, to lay out boards automatically, to check circuits for consistency, to guide wire-wrap machines, to specify combinational circuits and optimize them for programmed logic arrays (Chesson and Thompson). Without UCDS, significant inventions like Datakit, the 5620 Blit terminal, or the Belle chess machine would never have been built. UCDS appeared in only one manual, v7.

5. Communication

TSS (v2 page 64)

The members of the research group had no desire to isolate themselves from the rest of the Bell Labs computing community. Nor could they at first justify the purchase of equipment such as line printers and tape drives, which cost more than their whole computer. Thus, besides dial-up access, which was a *sine qua non*, communication with other machines was a necessity. A 2000bps link provided remote job entry to the GECOS system at the Bell Labs computer center (*opr*, Thompson, v2). GECOS guru Charlie Roberts contributed *tss* to exploit the link for remote login and interactive file transfer (v2). Similar programs to communicate with IBM mainframes followed.

Unlike most of the basic facilities of the system, which distilled years of well-established practice, computer-to-computer communication has been a subject of almost continuous experimentation. The need to connect to foreign systems exacerbates the problem of making a coherent model for communication. Ken Thompson more than once returned to ground zero, building experimental communication-based systems from scratch.

NFS (v7 pages 65-66), DCON (v8 page 67)

Sandy Fraser, aided by Jane Elliott, made our first local-area net, Spider, and its far-more-than-local-area successor based on the Datakit switch. Spider provided *nfs*, a remote network file store for a dozen minis (v7). Greg Chesson wrote remote-connection programs for direct machine-to-machine links. These programs, ultimately to become *dcon* (Chesson) and *rx* (Pike) for remote login and execution, were adapted to Datakit as soon as it became available. More recently they, and connection programs for other networks, have been reworked to exploit a general server mechanism by Ritchie and Presotto (v8).

Datakit connections sparked a version of the standard IO library that gave access to remote as well as local files (Fraser and Priscilla Lu). Ritchie's work on IO streams eventually made possible Weinberger's network file system, /n, which provided remote access through the kernel. Not to be confused with the central file server for Spider, Weinberger's file system is simply the union of all the files across all communicating machines. There is no manual page about how to use it because there is nothing to say. Remote file systems are "mounted" locally so that remote accesses look just like local accesses.

6. Security

SU (v1 page 68, v8 page 69)

Whether the system was actually run securely or not, considerable care has always been taken to assure that it is possible to do so. Permissions and Ritchie's patented set-userid mechanism were already supported in v1. From Cambridge, England, came the idea of password encryption that went into v3.

Trojan Horse tricks and countermeasures were discovered in an ongoing game that has been

recounted by Morris and Fred Grampp [4]. Notice, for example, the removal of *login*(1) to chapter 8 and the intrusion of `/etc/` into the synopsis for *su* (Grampp, v8). These fillips defeated the old chestnut of leaving programs named *login* or *su* lying around in hopes of capturing a password typed by an unwary system administrator. Other subtle features of the modern *su*: dot is excluded from the shell search path and the burglars' favorite shell variable `IFS` is reset.

CRYPT (v3 page 70, v9 page 71)

Morris's first file encrypter appeared in v3 with the explicit intent to stimulate code breaking experiments. Stimulate it did. Morris himself broke *crypt* by hand. Later Ritchie automated the cryptanalysis using a method of Jim Reeds (Berkeley). Completed with an editor interface, a new crypt went public in v7. It also succumbed to an attack by Reeds and Weinberger—and fortunately, too: more than one person who locked data in *crypt* and threw away the key has been rescued by code breakers.

But the still arduous process of code-breaking is not the easiest way to attack *crypt*. A simpler gambit is to catch a system administrator off guard and install a Trojan horse in *crypt* itself to snatch every new secret as it passes by. Thus the very presence of *crypt* may have just the opposite effect on security from what was intended.

Even if *crypt* were perfectly safe, it would be unwise to encrypt files of lasting value. It is too easy to lose the key, either inadvertently or deliberately. Consequently *crypt* has been demoted to the games chapter (Grampp, v9).

7. Curiosities

NUMBER (v6 page 72)

This filter that converts numbers to check-writing form was whipped up by Thompson (v6) to preprocess input to *speak*, which converted English to phonemes for a Votrax speech synthesizer (McIlroy, v3).* *Number* was the glue with which Thompson fashioned a talking desk calculator:

```
dc | number | speak
```

The talking calculator could also be reached from an audio shell, through which, with some effort, the whole system could be run from a Touch-Tone phone.

DSW (v1 page 73)

The nostalgically named *dsw* was a desperation tool designed to clean up files with unutterable names. This had been done on the PDP-7 by a program with console switch input; hence the sobriquet "delete from switches." It survived from v1 until it was displaced by `rm -i` (Ritchie, v7).

DD (v5 page 74)

Originally intended for converting files between the ASCII, little-endian, byte-stream world of DEC computers and the EBCDIC, big-endian, blocked world of IBM, *dd* was endowed with an appropriately bastard syntax (Thompson, v5). Pike has noted a cultural quirk. Much as families perpetuate the quaint sayings of children, users are wont to invoke *dd* with the JCL-ish formula, `dd if=input of=output`, or perhaps with `cat input | dd of=output`, but rarely with the elementary utterance `dd <input >output`.

---

* The UNIX lab was then the only place in the world where one could hear arbitrary text uttered by a machine on the spur of the moment. Visitors did a double-take upon being greeted by name. One hapless lounger listening to the fun from outside the lab door fled at a sudden sentence, "Stop messing around and get some work done."

FACED (v9 page 75)

Among the novelties inspired by bitmapped terminals was Pike's and Presottos's face server, which provides pictures of all users. The face server, like Weinberger's network file system and Killian's `/proc` directory of running core images, is a process that functions as a file system: it interprets file names upon *open*—in this case as connections to a remote repository—and delivers data upon *read*.

By far the most popular application of the face server is Luca Cardelli's *vismon*, which announces incoming messages with pictures of their senders.

Enigmatic images resembling that on the FACED page have frequently and inexplicably appeared on organization charts, posters, magazine covers, construction fences, and even a water tower.

8. Front matter

Titles and introductions (pages 76-96)

This chronological set of front matter comprises all title pages and prefaces plus samples of introductions and tables of contents.

In a bow to user-friendliness, Ritchie added "How to Get started" to the introduction for v3. At the typographic watershed where the manual was converted from *roff* to *troff* there appeared the thinnest, least forbidding UNIX Programmer's Manual of all time, v4 (pages 88-89). For v6 and v7 Lorinda Cherry prepared a pocket reference, familiarly known as the "Purple Card," which is not shown. Thompson gathered a set of *Documents for Use with UNIX*, also not shown, to accompany v6. Kernighan expanded the collection for the v7 manual, but nobody has had the ambition to modernize "Volume 2" since. Doug McIlroy compiled a glossary for the trade book form of v7 [2] and all later editions (page 94). Cherry prepared the topical table of contents for v9 (page 96).

Page 97 touches on software by Pike *et al.* for the Teletype 5620, the terminal of choice for v8 and v9. With a megabyte of memory and downloadable programs, the 5620 is a computer system in its own right. This fact is reflected in the organization of chapter 9 into subchapters parallel to the usual chapters: commands, system calls, subroutines, data layouts, games, etc. The 5620 software outweighs that of the original v1 in bulk, and offers a similar number of "primitive" functions. These facilities radically change the feel, but not the principles, of the UNIX system. Significant as it is, the 5620 software has been deliberately slighted in these readings about the common descent of UNIX, because the work is still tied to one kind of hardware and because it strikes off in new directions that more properly belong to a sequel.

Tables of contents (pages 98-114)

9. Acknowledgments

Many of the people mentioned have willingly provided fresh recollections, unearthed sources, and commented on drafts. I am grateful for the generous help of L. L. Cherry, J. H. Condon, R. J. Elliott, B. W. Kernighan, A. R. Koenig, L. E. McMahon, R. Pike, D. M. Ritchie, K. Thompson, P. J. Weinberger, and Norman Wilson. The selection of materials and whatever inaccuracies or oversights remain mine and mine alone.

11/3/71 CAT (I)

NAME cat -- concatenate and print

SYNOPSIS cat file$_1$ ...

DESCRIPTION cat reads each file in sequence and writes it on the standard output stream. Thus:

cat file

is about the easiest way to print a file. Also:

cat file1 file2 >file3

is about the easiest way to concatenate files.

If no input file is given cat reads from the standard input file.

FILES --

SEE ALSO pr, cp

DIAGNOSTICS none; if a file cannot be found it is ignored.

BUGS --

OWNER ken, dmr

11/3/71 CP (I)

NAME cp -- copy

SYNOPSIS cp $file_{11}$ $file_{12}$ $file_{21}$ $file_{22}$ ...

DESCRIPTION Files are taken in pairs; the first is opened for reading, the second created mode 17. Then the first is copied into the second.

FILES --

SEE ALSO cat, pr

DIAGNOSTICS Error returns are checked at every system call, and appropriate diagnostics are produced.

BUGS The second file should be created in the mode of the first.

A directory convention as used in mv should ba adopted to cp.

OWNER ken, dmr

NAME

cp – copy

SYNOPSIS

cp [–t] file1 file2

DESCRIPTION

The first file is copied onto the second. The mode and owner of the target file are preserved if it already existed; the mode of the source file is used otherwise.

If *file2* is a directory, then the target file is a file in that directory with the file-name of *file1*.

No one is quite sure what the flag –t does.

SEE ALSO

cat (I), pr (I), mv (I)

BUGS

Copying a file onto itself destroys its contents.

11/3/71 SORT (VI)

NAME sort -- sort a file

SYNOPSIS sort input output

DESCRIPTION sort will sort the input file and write the sorted file on the output file. Wide options are available on collating sequence and ignored characters.

FILES --

SEE ALSO --

DIAGNOSTICS --

BUGS --

OWNER dmr, ken

NAME

sort – sort or merge files

SYNOPSIS

sort [**−abdnrtx**] [*+pos* [*−pos*]] . . . [**−mo**] [name] . . .

DESCRIPTION

Sort sorts all the named files together and writes the result on the standard output. The name '−' means the standard input. The standard input is also used if no input file names are given. Thus *sort* may be used as a filter.

The default sort key is an entire line. Default ordering is lexicographic in ASCII collating sequence, except that lower-case letters are considered the same as the corresponding upper-case letters. Non-ASCII bytes are ignored. The ordering is affected by the flags **−abdnrt**, one or more of which may appear:

- **a** Do not map lower case letters.
- **b** Leading blanks (spaces and tabs) are not included in fields.
- **d** 'Dictionary' order: only letters, digits and blanks are significant in ASCII comparisons.
- **n** An initial numeric string, consisting of optional minus sign, digits and optionally included decimal point, is sorted by arithmetic value.
- **r** Reverse the sense of comparisons.
- **tx** Tab character between fields is *x*.

Selected parts of the line, specified by *+pos* and *−pos*, may be used as sort keys. *Pos* has the form *m.n*, where *m* specifies a number of fields to skip, and *n* a number of characters to skip further into the next field. A missing *.n* is taken to be 0. *+pos* denotes the beginning of the key; *−pos* denotes the first position after the key (end of line by default). The ordering rule may be overridden for a particular key by appending one or more of the flags **abdnr** to *+pos*.

When no tab character has been specified, a field consists of nonblanks and any preceding blanks. Under the **−b** flag, leading blanks are excluded from a field. When a tab character has been specified, a field is a string ending with a tab character.

When keys are specified, later keys are compared only when all earlier ones compare equal. Lines that compare equal are ordered with all bytes significant.

These flag arguments are also understood:

−m Merge only, the input files are already sorted.

−o The next argument is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs, except under the merge flag **−m**.

FILES

/usr/tmp/stm???

11/3/71 MAIL (I)

NAME mail -- send mail to another user

SYNOPSIS mail [letter person ...]

DESCRIPTION mail without an argument searches for a file called mailbox, prints it if present, and asks if it should be saved. If the answer is "y", the mail is renamed mail, otherwise it is deleted. The answer to the above question may be supplied in the letter argument.

When followed by the names of a letter and one or more people, the letter is appended to each person's mailbox. Each letter is preceded by the sender's name and a postmark.

A person is either the name of an entry in the directory /usr, in which case the mail is sent to /usr/person/mailbox, or the path name of a directory, in which case mailbox in that directory is used.

When a user logs in he is informed of the presence of mail.

FILES /etc/uids to map the sender's numerical user ID to name; mail and mailbox in various directories.

SEE ALSO init

DIAGNOSTICS "Who are you?" if the user cannot be identifed for some reason (a bug). "Cannot send to user" if mailbox cannot be opened.

BUGS --

OWNER ken

NAME

mail − send or receive mail among users

SYNOPSIS

mail person ...
mail [**−r**] [**−q**] [**−p**] [**−f** file]

DESCRIPTION

Mail with no argument prints a user's mail, message-by-message, in last-in, first-out order; the optional argument **−r** causes first-in, first-out order. If the **−p** flag is given, the mail is printed with no questions asked; otherwise, for each message, *mail* reads a line from the standard input to direct disposition of the message.

newline
: Go on to next message.

d
: Delete message and go on to the next.

p
: Print message again.

−
: Go back to previous message.

s [*file*] ...
: Save the message in the named *files* ('mbox' default).

w [*file*] ...
: Save the message, without a header, in the named *files* ('mbox' default).

m [*person*] ...
: Mail the message to the named *persons* (yourself is default).

EOT (control-D)
: Put unexamined mail back in the mailbox and stop.

q
: Same as EOT.

x
: Exit, without changing the mailbox file.

!command
: Escape to the Shell to do command.

?
: Print a command summary.

An interrupt stops the printing of the current letter. The optional argument **−q** causes *mail* to exit after interrupts without changing the mailbox.

When *persons* are named, *mail* takes the standard input up to an end-of-file (or a line with just '.') and adds it to each *person's* 'mail' file. The message is preceded by the sender's name and a postmark. Lines that look like postmarks are prepended with '>'. A *person* is usually a user name recognized by *login*(1). To denote a recipient on a remote system, prefix *person* by the system name and exclamation mark (see *uucp*(1)).

The **−f** option causes the named file, e.g. 'mbox', to be printed as if it were the mail file.

Each user owns his own mailbox, which is by default generally readable but not writable. The command does not delete an empty mailbox nor change its mode, so a user may make it unreadable if desired.

When a user logs in he is informed of the presence of mail.

FILES

/usr/spool/mail/*	mailboxes
/etc/passwd	to identify sender and locate persons
mbox	saved mail
/tmp/ma*	temp file

3/15/72 ECHO (I)

NAME echo -- echo arguments

SYNOPSIS echo [arg_1 ...]

DESCRIPTION echo writes all its arguments in order as a line on the standard output file. It is mainly useful for producing diagnostics in command files.

FILES --

SEE ALSO --

DIAGNOSTICS --

BUGS --

OWNER doug

NAME

grep – search a file for a pattern

SYNOPSIS

grep [**–v**] [**–l**] [**–n**] expression [input] [output]

DESCRIPTION

Grep will search the input file (standard input default) for each line containing the regular expression. Normally, each line found is printed on the output file (standard output default). If the **–v** flag is used, all lines but those matching are printed. If the **–l** flag is used, each line printed is preceded by its line number. If the **–n** flag is used, no lines are printed, but the number of lines that would normally have been printed is reported. If interrupt is hit, the number of lines searched is printed.

For a complete description of the regular expression, see ed(I). Care should be taken when using the characters **$ * [^ | ( )** and **** in the regular expression as they are also meaningful to the shell. (Precede them by ****)

SEE ALSO

ed(I), sh(I)

BUGS

Lines are limited to 512 characters; longer lines are truncated.

NAME sh -- shell (command interpreter)

SYNOPSIS sh [name [arg_1 ... [arg_9]]]

DESCRIPTION sh is the standard command interpreter. It is the program which reads and arranges the execution of the command lines typed by most users. It may itself be called as a command to interpret files of command lines. Before discussing the arguments to the shell used as a command, the structure of command lines themselves will be given.

Command lines are sequences of commands separated by command delimiters. Each command is a sequence of non-blank command arguments separated by blanks. The first argument specifies the name of a command to be executed. Except for certain types of special arguments discussed below, the arguments other than the command name are simply passed to the invoked command.

If the first argument represents the path name of an executable file, it is invoked; otherwise the string "/bin/" is prepended to the argument. (In this way the standard commands, which reside in "/bin", are found.) If this search too fails a diagnostic is printed.

The remaining non-special arguments are simply passed to the command without further interpretation by the shell.

There are three command delimiters: the new line, ";", and "&". The semicolon ";" specifies sequential execution of the commands so separated; that is,

```
coma; comb
```

causes the execution first of command coma, then of comb. The ampersand "&" causes simultaneous execution:

```
coma & comb
```

causes coma to be called, followed immediately by comb without waiting for coma to finish. Thus coma and comb execute simultaneously. As a special case,

```
coma &
```

causes coma to be executed and the shell immediately to request another command without waiting for coma.

Two characters cause the immediately following string to be interpreted as a special argument to the shell itself, not passed to the command. An argument of the form "<arg" causes the file arg to be used as the standard input file of the given command; an argument of the form ">arg" causes file "arg" to be used as the standard output file for the given command.

If any argument contains either of the characters "?" or "*", it is treated specially as follows. The current directory is searched for files which match the given argument. The character "*" in an argument matches any string of characters in a file name (including the null string); "?" matches any single character in a file name. Other argument characters match only the same character in the file name. For example, "*" matches all file names; "?" matches all one-character file names; "ab*.s" matches all file names beginning with "ab" and ending with ".s".

If the argument with "*" or "?" also contains a "/", a slightly different procedure is used: instead of the current directory, the directory used is the one obtained by taking the argument up to the last "/" before a "*" or "?". The matching process matches the remainder of the argument after this "/" against the files in the derived directory. For example: "/usr/dmr/a*.s" matches all files in directory "/usr/dmr" which begin with "a" and end with ".s".

In any event, a list of names is obtained which match the argument. This list is sorted into alphabetical order, and the resulting sequence of arguments replaces the single argument containing the "*" or "?". The same process is carried out for each argument with a "*" or "?" (the resulting lists are not merged) and finally the command is called with the resulting list of arguments.

For example: directory /usr/dmr contains the files a1.s, a2.s, ..., a9.s. From any directory, the command

```
as /usr/dmr/a?.s
```

calls as with arguments /usr/dmr/a1.s, /usr/dmr/a2.s, ... /usr/dmr/a9.s in that order.

The character "\" causes the immediately following character to lose any special meaning it may have to the shell; in this way "<", ">", and other characters meaningful to the shell may be passed as part of arguments. A special case of

this feature allows the continuation of commands onto more than one line: a new-line preceded by "\" is translated into a blank.

Sequences of characters enclosed in double (") or single (') quotes are also taken literally.

When the shell is invoked as a command, it has additional string processing capabilities. Recall that the form in which the whell is invoked is

sh [name [arg_1 ... [arg_9]]]

The name is the name of a file which will be read and interpreted. If not given, this subinstance of the shell will continue to read the standard input file.

In the file, character sequences of the form "$n", where n is a digit 0, ..., 9, are replaced by the nth argument to the invocation of the shell (arg_n). "$0" is replaced by name.

An end-of-file in the shell's input causes it to exit. A side effect of this fact means that the way to log out from UNIX is to type an end of file.

FILES /etc/glob

SEE ALSO [reference], which gives the theory of operation of the shell.

DIAGNOSTICS "?", in case of any difficulty. The most common problem is inability to find the given command. Others: input file ("<") cannot be found; no more processes can be created (this will alleviate itself with the passage of time). Note that no diagnostic is given for inability to create an output (">") file; the standard output file has already been closed when the condition is discovered and there is no place to write the diagnostic.

If a "*" or "?" is used, the glob routine is invoked; it types "No command" if it cannot find the given command, and "No match" if there were no files which matched an argument with "?" or "*".

BUGS Better diagnostics should be provided. If a "*" or "?" is used, the command must be in /bin. (Not, for example, in the user's directory.) This is actually a glob bug.

NAME sh -- shell (command interpreter)

SYNOPSIS sh [name [arg_1 ... [arg_9]]]

DESCRIPTION

sh is the standard command interpreter. It is the program which reads and arranges the execution of the command lines typed by most users. It may itself be called as a command to interpret files of commands. Before discussing the arguments to the shell used as a command, the structure of command lines themselves will be given.

Command lines

Command lines are sequences of commands separated by command delimiters. Each command is a sequence of non-blank command arguments separated by blanks. The first argument specifies the name of a command to be executed. Except for certain types of special arguments discussed below, the arguments other than the command name are passed without interpretation to the invoked command.

If the first argument is the name of an executable file, it is invoked; otherwise the string "/bin/" is prepended to the argument. (In this way most standard commands, which reside in "/bin", are found.) If no such command is found, the string "/usr" is further prepended (to give "/usr/bin/command") and another attempt is made to execute the resulting file. (Certain "overflow" commands live in "/usr/bin".) If the "/usr/bin" file exists, but is not executable, it is used by the shell as a command file. That is to say it is executed as though it were typed from the console. If all attempts fail, a diagnostic is printed.

The remaining non-special arguments are simply passed to the command without further interpretation by the shell.

Command delimiters

There are three command delimiters: the new-line, ";", and "&". The semicolon ";" specifies sequential execution of the commands so separated; that is,

```
coma; comb
```

causes the execution first of command coma, then of comb. The ampersand "&" causes simultaneous execution:

```
coma & comb
```

causes coma to be called, followed immediately by comb without waiting for coma to finish. Thus coma and comb execute simultaneously. As a special case,

```
coma &
```

causes coma to be executed and the shell immediately to request another command without waiting for coma.

Termination Reporting

If a command (not followed by "&") terminates abnormally, a message is printed. (All terminations other than exit and interrupt are considered abnormal.) The following is a list of the abnormal termination messages:

```
Bus error
Trace/BPT trap
Illegal instruction
IOT trap
Power fail trap
EMT trap
Bad system call
Quit
PIR trap
Floating exception
Memory violation
Killed
User I/O
Error
```

If a core image is produced, " — Core dumped" is appended to the appropriate message.

Redirection of I/O

Three character sequences cause the immediately following string to be interpreted as a special argument to the shell itself, not passed to the command.

An argument of the form "<arg" causes the file arg to be used as the standard input file of the given command.

An argument of the form ">arg" causes file "arg" to be used as the standard output file for the given command. "Arg" is created if it did not exist, and in any case is truncated at the outset.

An argument of the form ">>arg" causes file "arg" to be used as the standard output for the given command. If "arg" did not exist, it is created; if it did exist, the command output is appended to the file.

Pipes and Filters

A pipe is a channel such that information can be written into one end of the pipe by one program, and read at the other end by another program. (See pipe (II)). A filter is a program which reads the standard input file, performs some transformation, and writes the result on the standard output file. By extending the syntax used for redirection of I/O, a command line can specify that the

output produced by a command be passed via a pipe through another command which acts as a filter. For example:

```
command >filter>
```

More generally, special arguments of the form

$>f_1>f_2>...>$

specify that output is to be passed successively through the filters f_1, f_2, ..., and end up on the standard output stream. By saying instead

$>f_1>f_2>...>$file

the output finally ends up in file. (The last ">" could also have been a ">>" to specify concatenation onto the end of file.)

In exactly analogous manner input filtering can be specified via one of

$<f_1<f_2<...<$
$<f_1<f_2<...<$file

Both input and output filtering can be specified in the same command, though not in the same special argument.

For example:

```
ls >pr>
```

produces a listing of the current directory with page headings, while

```
ls >pr>xx
```

puts the paginated listing into the file xx.

If any of the filters needs arguments, quotes can be used to prevent the required blank characters from violating the blankless syntax of filters. For example:

```
ls >"pr -h 'My directory'">
```

uses quotes twice, once to protect the entire pr command, once to protect the heading argument of pr. (Quotes are discussed fully below.)

Generation of argument lists

If any argument contains any of the characters "?", "*" or '[', it is treated specially as follows. The current directory is searched for files which match the given argument.

The character "*" in an argument matches any string of characters in a file name (including the null string).

The character "?" matches any single character in a file name.

Square brackets "[...]" specify a class of characters which matches any single file-name character in the class. Within the brackets, each ordinary character is taken to be a member of the class. A pair of characters separated by "-" places in the class each character lexically greater than or equal to the first and less than or equal to the second member of the pair.

Other characters match only the same character in the file name.

For example, "*" matches all file names; "?" matches all one-character file names; "[ab]*.s" matches all file names beginning with "a" or "b" and ending with ".s"; "?[zi-m]" matches all two-character file names ending with "z" or the letters "i" through "m".

If the argument with "*" or "?" also contains a "/", a slightly different procedure is used: instead of the current directory, the directory used is the one obtained by taking the argument up to the last "/" before a "*" or "?". The matching process matches the remainder of the argument after this "/" against the files in the derived directory. For example: "/usr/dmr/a*.s" matches all files in directory "/usr/dmr" which begin with "a" and end with ".s".

In any event, a list of names is obtained which match the argument. This list is sorted into alphabetical order, and the resulting sequence of arguments replaces the single argument containing the "*", "[", or "?". The same process is carried out for each argument (the resulting lists are not merged) and finally the command is called with the resulting list of arguments.

For example: directory /usr/dmr contains the files a1.s, a2.s, ..., a9.s. From any directory, the command

```
as /usr/dmr/a?.s
```

calls as with arguments /usr/dmr/a1.s, /usr/dmr/a2.s, ... /usr/dmr/a9.s in that order.

Quoting

The character "\" causes the immediately following character to lose any special meaning it may have to the shell; in this way "<", ">", and other characters meaningful to the shell may be passed as part of arguments. A special case of this feature allows the continuation of

commands onto more than one line: a new-line preceded by "\" is translated into a blank.

Sequences of characters enclosed in double (") or single (') quotes are also taken literally.

Argument passing

When the shell is invoked as a command, it has additional string processing capabilities. Recall that the form in which the shell is invoked is

sh [name [arg_1 ... [arg_9]]]

The name is the name of a file which will be read and interpreted. If not given, this subinstance of the shell will continue to read the standard input file.

In command lines in the file (not in command input), character sequences of the form "$n", where n is a digit 0, ..., 9, are replaced by the nth argument to the invocation of the shell (arg_n). "$0" is replaced by name.

End of file

An end-of-file in the shell's input causes it to exit. A side effect of this fact means that the way to log out from UNIX is to type an end of file.

Special commands

Two commands are treated specially by the shell.

"Chdir" is done without spawning a new process by executing the sys chdir primitive.

"Login" is done by executing /bin/login without creating a new process.

These peculiarities are inexorably imposed upon the shell by the basic structure of the UNIX process control system. It is a rewarding exercise to work out why.

Command file errors; interrupts

Any shell-detected error, or an interrupt signal, during the execution of a command file causes the shell to cease execution of that file.

FILES /etc/glob, which interprets "*", "?", and "[".

SEE ALSO "The UNIX Time-sharing System", which gives the theory of operation of the shell.

DIAGNOSTICS
"Input not found", when a command file is specified which

cannot be read;
"Arg count", if the number of arguments to the chdir pseudo-command is not exactly 1, or if "*", "?", or "[" is used inappropriately;
"Bad directory", if the directory given in "chdir" cannot be switched to;
"Try again", if no new process can be created to execute the specified command;
"", imbalance", if single or double quotes are not matched;
"Input file", if an argument after "<" cannot be read;
"Output file", if an argument after ">" or ">>" cannot be written (or created);
"Command not found", if the specified command cannot be executed.
"No match", if no arguments are generated for a command which contains "*", "?", or "[".
Termination messages described above.

BUGS If any argument contains a quoted "*", "?", or "[", then all instances of these characters must be quoted. This is because sh calls the glob routine whenever an unquoted "*", "?", or "[" is noticed; the fact that other instances of these characters occurred quoted is not noticed by glob.

When output is redirected, particularly through a filter, diagnostics tend to be sent down the pipe and are sometimes lost altogether.

NAME

sh – shell (command interpreter)

SYNOPSIS

sh [name [arg1 ... [arg9]]]

DESCRIPTION

Sh is the standard command interpreter. It is the program which reads and arranges the execution of the command lines typed by most users. It may itself be called as a command to interpret files of commands. Before discussing the arguments to the Shell used as a command, the structure of command lines themselves will be given.

Commands. Each command is a sequence of non-blank command arguments separated by blanks. The first argument specifies the name of a command to be executed. Except for certain types of special arguments discussed below, the arguments other than the command name are passed without interpretation to the invoked command.

If the first argument is the name of an executable file, it is invoked; otherwise the string '/bin/' is prepended to the argument. (In this way most standard commands, which reside in '/bin', are found.) If no such command is found, the string '/usr' is further prepended (to give '/usr/bin/command') and another attempt is made to execute the resulting file. (Certain lesser-used commands live in '/usr/bin'.) If the '/usr/bin' file exists, but is not executable, it is used by the Shell as a command file. That is to say it is executed as though it were typed from the console. If all attempts fail, a diagnostic is printed.

Command lines. One or more commands separated by '|' or '^' constitute a *pipeline*. The standard output of each command but the last in a pipeline is taken as the standard input of the next command. Each command is run as a separate process, connected by pipes (see pipe(II)) to its neighbors. A command line contained in parentheses '( )' may appear in place of a simple command as an element of a pipeline.

A *command line* consists of one or more pipelines separated, and perhaps terminated by ';' or '&'. The semicolon designates sequential execution. The ampersand causes the preceding pipeline to be executed without waiting for it to finish. The process id of such a pipeline is reported, so that it may be used if necessary for a subsequent *wait* or *kill*.

Termination Reporting. If a command (not followed by '&') terminates abnormally, a message is printed. (All terminations other than exit and interrupt are considered abnormal.) Termination reports for commands followed by '&' are given upon receipt of the first command subsequent to the termination of the command, or when a *wait* is executed. The following is a list of the abnormal termination messages:

Bus error
Trace/BPT trap
Illegal instruction
IOT trap
EMT trap
Bad system call
Quit
Floating exception
Memory violation
Killed

If a core image is produced, '– Core dumped' is appended to the appropriate message.

Redirection of I/O. There are three character sequences that cause the immediately following string to be interpreted as a special argument to the Shell itself. Such an argument may appear anywhere among the arguments of a simple command, or before or after a parenthesized command list, and is associated with that command or command list.

An argument of the form '<arg' causes the file 'arg' to be used as the standard input file of the associated command.

An argument of the form '>arg' causes file 'arg' to be used as the standard output file for the associated command. 'Arg' is created if it did not exist, and in any case is truncated at the outset.

An argument of the form '>>arg' causes file 'arg' to be used as the standard output for the associated command. If 'arg' did not exist, it is created; if it did exist, the command output is appended to the file.

For example, either of the command lines

```
ls >junk; cat tail >>junk
( ls; cat tail ) >junk
```

creates, on file 'junk', a listing of the working directory, followed immediately by the contents of file 'tail'.

Either of the constructs '>arg' or '>>arg' associated with any but the last command of a pipeline is ineffectual, as is '<arg' in any but the first.

Generation of argument lists. If any argument contains any of the characters '?', '*' or '[', it is treated specially as follows. The current directory is searched for files which *match* the given argument.

The character '*' in an argument matches any string of characters in a file name (including the null string).

The character '?' matches any single character in a file name.

Square brackets '[...]' specify a class of characters which matches any single file-name character in the class. Within the brackets, each ordinary character is taken to be a member of the class. A pair of characters separated by '-' places in the class each character lexically greater than or equal to the first and less than or equal to the second member of the pair.

Other characters match only the same character in the file name.

For example, '*' matches all file names; '?' matches all one-character file names; '[ab]* s' matches all file names beginning with 'a' or 'b' and ending with '.s'; '?[zi-m]' matches all two-character file names ending with 'z' or the letters 'i' through 'm'.

If the argument with '*' or '?' also contains a '/', a slightly different procedure is used: instead of the current directory, the directory used is the one obtained by taking the argument up to the last '/' before a '*' or '?'. The matching process matches the remainder of the argument after this '/' against the files in the derived directory. For example: '/usr/dmr/a*.s' matches all files in directory '/usr/dmr' which begin with 'a' and end with '.s'.

In any event, a list of names is obtained which match the argument. This list is sorted into alphabetical order, and the resulting sequence of arguments replaces the single argument containing the '*', '[', or '?'. The same process is carried out for each argument (the resulting lists are *not* merged) and finally the command is called with the resulting list of arguments.

For example: directory /usr/dmr contains the files a1.s, a2.s, ..., a9.s. From any directory, the command

```
as /usr/dmr/a?.s
```

calls *as* with arguments /usr/dmr/a1.s, /usr/dmr/a2.s, ... /usr/dmr/a9.s in that order.

Quoting. The character '\' causes the immediately following character to lose any special meaning it may have to the Shell; in this way '<', '>', and other characters meaningful to the Shell may be passed as part of arguments. A special case of this feature allows the continuation of commands onto more than one line: a new-line preceded by '\' is translated into a blank.

Sequences of characters enclosed in double (") or single (') quotes are also taken literally. For example:

```
ls | pr -h "My directory"
```

causes a directory listing to be produced by *ls*, and passed on to *pr* to be printed with the heading 'My directory'. Quotes permit the inclusion of blanks in the heading, which is a single argument

to *pr*.

Argument passing. When the Shell is invoked as a command, it has additional string processing capabilities. Recall that the form in which the Shell is invoked is

sh [name [arg1 ... [arg9]]]

The *name* is the name of a file which will be read and interpreted. If not given, this subinstance of the Shell will continue to read the standard input file.

In command lines in the file (not in command input), character sequences of the form '$n', where *n* is a digit, are replaced by the *n*th argument to the invocation of the Shell (argn). '$0' is replaced by *name*.

End of file. An end-of-file in the Shell's input causes it to exit. A side effect of this fact means that the way to log out from UNIX is to type an EOT.

Special commands. The following commands are treated specially by the Shell.

chdir is done without spawning a new process by executing *sys chdir* (II).

login is done by executing /bin/login without creating a new process.

wait is done without spawning a new process by executing *sys wait* (II).

shift is done by manipulating the arguments to the Shell.

':' is simply ignored.

Command file errors; interrupts. Any Shell-detected error, or an interrupt signal, during the execution of a command file causes the Shell to cease execution of that file.

Process that are created with a '&' ignore interrupts. Also if such a process has not redirected its input with a '<', its input is automatically redirected to the zero length file /dev/null.

FILES

/etc/glob, which interprets '*', '?', and '['.
/dev/null as a source of end-of-file.

SEE ALSO

'The UNIX Time-sharing System', which gives the theory of operation of the Shell.
chdir(I), login(I), wait(I), shift(I)

BUGS

When output is redirected, particularly to make a multicommand pipeline, diagnostics tend to be sent down the pipe and are sometimes lost altogether. Not all components of a pipeline swawned with '&' ignore interrupts.

NAME goto -- command transfer

SYNOPSIS goto label

DESCRIPTION goto is only allowed when the Shell is taking commands from a file. The file is searched (from the beginning) for a line beginning with ":" followed by one or more spaces followed by the label. If such a line is found, the goto command returns. Since the read pointer in the command file points to the line after the label, the effect is to cause the Shell to transfer to the labelled line.

":" is a do-nothing command that only serves to place a label.

FILES --

SEE ALSO sh(I), :(I)

DIAGNOSTICS "goto error", if the input file is a typewriter; "label not found".

BUGS --

OWNER dmr

3/15/72 : (I)

NAME : -- place a label

SYNOPSIS : [label]

DESCRIPTION : does nothing. Its only function is to place a label for the goto command. : is a command so the Shell doesn't have to be fixed to ignore lines with :'s.

FILES --

SEE ALSO goto(I)

DIAGNOSTICS --

BUGS --

OWNER dmr

11/3/71 SYS STAT (II)

NAME stat -- get file status

SYNOPSIS
```
sys     stat; name; buf / stat = 18.
```

DESCRIPTION

name points to a null-terminated string naming a file; buf is the address of a 34(10) byte buffer into which information is placed concerning the file. It is unnecessary to have any permissions at all with respect to the file, but all directories leading to the file must be readable.

After stat, buf has the following format:

```
buf, +1             i-number
+2,+3               flags (see below)
+4                  number of links
+5                  user ID of owner
+6,+7               size in bytes
+8,+9               first indirect block or contents block
...
+22,+23             eighth indirect block or contents block
+24,+25,+26,+27     creation time
+28,+29,+30,+31     modification time
+32,+33             unused
```

The flags are as follows:

```
100000  used (always on)
040000  directory
020000  file has been modified (always on)
010000  large file
000040  set user ID
000020  executable
000010  read, owner
000004  write, owner
000002  read, non-owner
000001  write, non-owner
```

FILES --

SEE ALSO fstat

DIAGNOSTICS Error bit (c-bit) is set if the file cannot be found.

BUGS The format is going to change someday.

OWNER ken, dmr

NAME

stat – get file status

SYNOPSIS

(stat = 18.)
sys stat; name; buf

stat(name, buf)
char *name;
struct inode *buf;

DESCRIPTION

Name points to a null-terminated string naming a file; *buf* is the address of a 36(10) byte buffer into which information is placed concerning the file. It is unnecessary to have any permissions at all with respect to the file, but all directories leading to the file must be readable. After *stat, buf* has the following structure (starting offset given in bytes):

```
struct {
 char        minor;        /* +0: minor device of i-node */
 char        major;        /* +1: major device */
 int         inumber       /* +2 */
 int         flags;        /* +4: see below */
 char        nlinks;       /* +6: number of links to file */
 char        uid;          /* +7: user ID of owner */
 char        gid;          /* +8: group ID of owner */
 char        size0;        /* +9: high byte of 24-bit size */
 int         size1;        /* +10: low word of 24-bit size */
 int         addr[8];      /* +12: block numbers or device number */
 int         actime[2];    /* +28: time of last access */
 int         modtime[2];   /* +32: time of last modification */
};
```

The flags are as follows:

```
100000   i-node is allocated
060000   2-bit file type:
     000000   plain file
     040000   directory
     020000   character-type special file
     060000   block-type special file.
010000   large file
004000   set user-ID on execution
002000   set group-ID on execution
000400   read (owner)
000200   write (owner)
000100   execute (owner)
000070   read, write, execute (group)
000007   read, write, execute (others)
```

SEE ALSO

stat(I), fstat(II), fs(V)

DIAGNOSTICS

Error bit (c-bit) is set if the file cannot be found. From C, a −1 return indicates an error.

NAME

stat, fstat − get file status

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>

stat(name, buf)
char *name;
struct stat *buf;

fstat(fildes, buf)
struct stat *buf;
```

DESCRIPTION

Stat obtains detailed information about a named file. *Fstat* obtains the same information about an open file known by the file descriptor from a successful *open*, *creat*, *dup* or *pipe*(2) call.

Name points to a null-terminated string naming a file; *buf* is the address of a buffer into which information is placed concerning the file. It is unnecessary to have any permissions at all with respect to the file, but all directories leading to the file must be searchable. The layout of the structure pointed to by buf as defined in <*stat.h*> is given below. *St_mode* is encoded according to the '#define' statements.

```
struct  stat
{
        dev_t   st_dev;
        ino_t   st_ino;
        unsigned short st_mode;
        short   st_nlink;
        short   st_uid;
        short   st_gid;
        dev_t   st_rdev;
        off_t   st_size;
        time_t  st_atime;
        time_t  st_mtime;
        time_t  st_ctime;
};

#define S_IFMT      0170000     /* type of file */
#define     S_IFDIR     0040000     /* directory */
#define     S_IFCHR     0020000     /* character special */
#define     S_IFBLK     0060000     /* block special */
#define     S_IFREG     0100000     /* regular */
#define     S_IFMPC     0030000     /* multiplexed char special */
#define     S_IFMPB     0070000     /* multiplexed block special */
#define S_ISUID     0004000     /* set user id on execution */
#define S_ISGID     0002000     /* set group id on execution */
#define S_ISVTX     0001000     /* save swapped text even after use */
#define S_IREAD     0000400     /* read permission, owner */
#define S_IWRITE    0000200     /* write permission, owner */
#define S_IEXEC     0000100     /* execute/search permission, owner */
```

The mode bits 0000070 and 0000007 encode group and others permissions (see *chmod*(2)). The defined types, *ino_t*, *off_t*, *time_t*, name various width integer values; *dev_t* encodes major and minor device numbers; their exact definitions are in the include file <sys/types.h> (see *types*(5).

When *fildes* is associated with a pipe, *fstat* reports an ordinary file with restricted permissions. The size is the number of bytes queued in the pipe.

st_atime is the file was last read. For reasons of efficiency, it is not set when a directory is searched, although this would be more logical. *st_mtime* is the time the file was last written or created. It is not set by changes of owner, group, link count, or mode. *st_ctime* is set both both by writing and changing the i-node.

SEE ALSO

ls(1), filsys(5)

DIAGNOSTICS

Zero is returned if a status is available; −1 if the file cannot be found.

ASSEMBLER

(stat = 18.)
sys stat; name; buf

(fstat = 28.)
(file descriptor in r0)
sys fstat; buf

NAME pipe -- create a pipe

SYNOPSIS
```
sys pipe        / pipe = 42.; not in assembler
(file descriptor in r0)
```

DESCRIPTION The pipe system call creates an I/O mechanism called a pipe. The file descriptor returned can be used in both read and write operations. When the pipe is written, the data is buffered up to 504 bytes at which time the writing process is suspended. A read on the pipe will pick up the buffered data.

It is assumed that after the pipe has been set up, two (or more) cooperating processes (created by subsequent fork calls) will pass data through the pipe with read and write calls.

The shell has a syntax to set up a linear array of processes connected by pipes.

Read calls on an empty pipe (no buffered data) with only one end (no synonymous file descriptors resulting from fork or dup) return an end-of-file. Write calls under similar conditions are ignored.

SEE ALSO sh(I), read(II), write(II), fork(II)

DIAGNOSTICS The error bit (c-bit) is set if 10 files are already open.

BUGS --

NAME intr -- set interrupt handling

SYNOPSIS sys intr; arg / intr = 27.

DESCRIPTION When arg is 0, interrupts (ASCII DELETE) are ignored. When arg is 1, interrupts cause their normal result, that is, force an exit. When arg is a location within the program, control is transferred to that location when an interrupt occurs.

After an interrupt is caught, it is possible to resume execution by means of an rti instruction; however, great care must be exercised, since all I/O is terminated abruptly upon an interrupt. In particular, reads of the typewriter tend to return with 0 characters read, thus simulating an end of file.

FILES --

SEE ALSO quit

DIAGNOSTICS --

BUGS It should be easier to resume after an interrupt, but I don't know how to make it work.

OWNER ken, dmr

NAME

signal – catch or ignore signals

SYNOPSIS

(signal = 48.)
sys signal; sig; value

signal(sig, func)
int (*func)();

DESCRIPTION

When the signal defined by *sig* is sent to the current process, it is to be treated according to *value*. The following is the list of signals:

1	hangup
2	interrupt
3*	quit
4*	illegal instruction
5*	trace trap
6*	IOT instruction
7*	EMT instruction
8*	floating point exception
9	kill (cannot be caught or ignored)
10*	bus error
11*	segmentation violation
12*	bad argument to sys call

If *value* is 0, the default system action applies to the signal. This is processes termination with or without a core dump. If *value* is odd, the signal is ignored. Any other even *value* specifies an address in the process where an interrupt is simulated. An RTI instruction will return from the interrupt. As a signal is caught, it is reset to 0. Thus if it is desired to catch every such signal, the catching routine must issue another *signal* call.

The starred signals in the list above cause core images if not caught and not ignored. In C, if *func* is 0 or 1, the action is as described above. If *func* is even, it is assumed to be the address of a function entry point. When the signal occurs, the function will be called. A return from the function will simulate the RTI.

After a *fork*, the child inherits all signals. The *exec* call resets all caught signals to default action.

SEE ALSO

kill (I, II)

DIAGNOSTICS

The error bit (c-bit) is set if the given signal is out of range. In C, a −1 indicates an error; 0 indicates success.

6/12/72 STTY (II)

NAME stty — set mode of typewriter

SYNOPSIS
```
(file descriptor in r0)
sys stty; arg  / stty = 31.
...
arg: dcrsr; dcpsr; mode
```

DESCRIPTION

stty sets mode bits for a typewriter whose file descriptor is passed in r0. First, the system delays until the typewriter is quiescent. Then, the argument dcrsr is placed into the typewriter's receiver control and status register, and dcpsr is placed in the transmitter control and status register. The DC-11 manual must be consulted for the format of these words. For the purpose of this call, the most important rôle of these arguments is to adjust to the speed of the typewriter.

The mode arguments contains several bits which determine the system's treatment of the typewriter:

200	even parity allowed on input (e. g. for m37s)
100	odd parity allowed on input
040	raw mode: wake up on all characters
020	map CR into LF; echo LF or CR as LF-CR
010	echo (full duplex)
004	map upper case to lower on input (e. g. M33)
002	echo and print tabs as spaces
001	inhibit all function delays (e. g. CRTs)

Characters with the wrong parity, as determined by bits 200 and 100, are ignored.

In raw mode, every character is passed back immediately to the program. No erase or kill processing is done; the end-of-file character (EOT), the interrupt character (DELETE) and the quit character (FS) are not treated specially.

Mode 020 causes input carriage returns to be turned into new-lines; input of either CR or LF causes LF-CR both to be echoed (used for GE TermiNet 300's and other terminals without the newline function).

Additional bits in the high order byte of the mode argument are used to indicate that the terminal is an IBM 2741 and to specify 2741 modes. These mode bits are:

4000 use correspondence code conversion on input (currently ignored)

Normal input and output code conversion for 2741s is EBCDIC (e. g. 963 ball and corresponding keyboard). The presence of the transmit interrupt feature permits the system to do read-ahead while no output is in progress. In 2741 mode, the low order bits 331 are ignored.

FILES --

SEE ALSO stty(I), gtty(II)

DIAGNOSTICS The error bit (c-bit) is set if the file descriptor does not refer to a typewriter.

BUGS This call should be used with care. It is all too easy to turn off your typewriter.

OWNER ken, dmr

IOCTL (2)

NAME

ioctl, stty, gtty – control device

SYNOPSIS

```
#include <sgtty.h>

ioctl(fildes, request, argp)
struct sgttyb *argp;

stty(fildes, argp)
struct sgttyb *argp;

gtty(fildes, argp)
struct sgttyb *argp;
```

DESCRIPTION

Ioctl performs a variety of functions on character special files (devices). The writeups of various devices in section 4 discuss how *ioctl* applies to them.

For certain status setting and status inquiries about terminal devices, the functions *stty* and *gtty* are equivalent to

```
ioctl(fildes, TIOCSETP, argp)
ioctl(fildes, TIOCGETP, argp)
```

respectively; see *tty*(4).

The following two calls, however, apply to any open file:

```
ioctl(fildes, FIOCLEX, NULL);
ioctl(fildes, FIONCLEX, NULL);
```

The first causes the file to be closed automatically during a successful *exec* operation; the second reverses the effect of the first.

SEE ALSO

stty(1), tty(4), exec(2)

DIAGNOSTICS

Zero is returned if the call was successful; −1 if the file descriptor does not refer to the kind of file for which it was intended.

BUGS

Strictly speaking, since *ioctl* may be extended in different ways to devices with different properties, *argp* should have an open-ended declaration like

```
union { struct sgttyb ...; ... } *argp;
```

The important thing is that the size is fixed by 'struct sgttyb'.

ASSEMBLER

```
(ioctl = 54.)
sys ioctl; fildes; request; argp

(stty = 31.)
(file descriptor in r0)
stty; argp

(gtty = 32.)
(file descriptor in r0)
sys gtty; argp
```

NAME

printf – formatted print

SYNOPSIS

printf(format, arg$_1$, ...);
char *format;

DESCRIPTION

Printf converts, formats, and prints its arguments after the first under control of the first argument. The first argument is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of the next successive argument to *printf*.

Each conversion specification is introduced by the character %. Following the %, there may be

- an optional minus sign '–' which specifies *left adjustment* of the converted argument in the indicated field;
- an optional digit string specifying a *field width;* if the converted argument has fewer characters than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width;
- an optional period "." which serves to separate the field width from the next digit string;
- an optional digit string *(precision)* which specifies the number of digits to appear after the decimal point, for e- and f-conversion, or the maximum number of characters to be printed from a string;
- a character which indicates the type of conversion to be applied.

The conversion characters and their meanings are

d The argument is converted to decimal notation.

o The argument is converted to octal notation. "0" will always appear as the first digit.

f The argument is converted to decimal notation in the style "[–]ddd.ddd" where the number of d's after the decimal point is equal to the precision specification for the argument. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed. The argument should be *float* or *double*.

e The argument is converted in the style "[–]d.ddde±dd" where there is one digit before the decimal point and the number after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced. The argument should be a *float* or *double* quantity.

c The argument character or character-pair is printed if non-null.

s The argument is taken to be a string (character pointer) and characters from the string are printed until a null character or until the number of characters indicated by the precision specification is reached; however if the precision is 0 or missing all characters up to a null are printed.

l The argument is taken to be an unsigned integer which is converted to decimal and printed (the result will be in the range 0 to 65535).

If no recognizable character appears after the %, that character is printed; thus % may be printed by use of the string %%. In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width. Characters generated by *printf* are printed by calling *putchar*.

SEE ALSO

putchar (III)

NAME putc, putw, fcreat, flush -- buffered output

SYNOPSIS

```
mov     $filename,r0
jsr     r5,fcreat; iobuf

(get byte in r0)
jsr     r5,putc; iobuf

(get word in r0)
jsr     r5,putw; iobuf

jsr     r5,flush; iobuf
```

DESCRIPTION

fcreat creates the given file (mode 17) and sets up the buffer iobuf (size 134(10) bytes); putc and putw write a byte or word respectively onto the file; flush forces the contents of the buffer to be written, but does not close the file. The format of the buffer is:

```
iobuf:  .=.+2           / file descriptor
        .=.+2           / characters unused in buffer
        .=.+2           / ptr to next free character
        .=.+128.        / buffer
```

fcreat sets the error bit (c-bit) if the file creation failed; none of the other routines return error information.

Before terminating, a program should call flush to force out the last of the output.

The user must supply iobuf, which should begin on a word boundary.

FILES kept in /etc/liba.a

SEE ALSO sys creat; sys write; getc, getw, fopen

DIAGNOSTICS error bit possible on fcreat call

BUGS buffers should be changed to 512 bytes.

OWNER dmr

NAME

stdio − standard buffered input/output package

SYNOPSIS

```
#include <stdio.h>

FILE *stdin;
FILE *stdout;
FILE *stderr;
```

DESCRIPTION

The functions described in Sections 3S constitute an efficient user-level buffering scheme. The in-line macros *getc* and *putc*(3) handle characters quickly. The higher level routines *gets*, *fgets*, *scanf*, *fscanf*, *fread*, *puts*, *fputs*, *printf*, *fprintf*, *fwrite* all use *getc* and *putc*; they can be freely intermixed.

A file with associated buffering is called a *stream*, and is declared to be a pointer to a defined type **FILE**. *Fopen*(3) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. There are three normally open streams with constant pointers declared in the include file and associated with the standard open files:

stdin	standard input file
stdout	standard output file
stderr	standard error file

A constant 'pointer' **NULL** (0) designates no stream at all.

An integer constant **EOF** (−1) is returned upon end of file or error by integer functions that deal with streams.

Any routine that uses the standard input/output package must include the header file <stdio.h> of pertinent macro definitions. The functions and constants mentioned in sections labeled 3S are declared in the include file and need no further declaration. The constants, and the following 'functions' are implemented as macros; redeclaration of these names is perilous: *getc*, *getchar*, *putc*, *putchar*, *feof*, *ferror*, *fileno*.

SEE ALSO

open(2), close(2), read(2), write(2)

DIAGNOSTICS

The value **EOF** is returned uniformly to indicate that a **FILE** pointer has not been initialized with *fopen*, input (output) has been attempted on an output (input) stream, or a **FILE** pointer designates corrupt or otherwise unintelligible **FILE** data.

NAME

cc -- C compiler

SYNOPSIS

cc [-c] sfile$_1$.c ... ofile$_1$ ...

DESCRIPTION

cc is the UNIX C compiler. It accepts three types of arguments:

Arguments whose names end with ".c" are assumed to be C source programs; they are compiled, and the object program is left on the file sfile$_1$.o (i.e., the file whose name is that of the source with ".o" substituted for ".c").

Other arguments (except for "-c") are assumed to be either loader flag arguments, or C-compatible object programs, typically produced by an earlier cc run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name a.out.

The "-c" argument suppresses the loading phase, as does any syntax error in any of the routines being compiled.

FILES

file.c	input file
a.out	loaded output
c.tmp	temporary (deleted)
/sys/c/nc	compiler
/usr/lib/crt0.o	runtime startoff
/usr/lib/libc.a	builtin functions, etc.
/usr/lib/liba.a	system library

SEE ALSO

C reference manual (in preparation), bc(VI)

DIAGNOSTICS

Diagnostics are intended to be self-explanatory.

BUGS

—

OWNER

dmr

NAME

yacc — yet another compiler compiler

SYNOPSIS

/crp/scj/yacc [<grammar]

DESCRIPTION

Yacc converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The tables are provided in readable form on the standard output and in b-compiler format on file actn.b; the program /crp/scj/bpar.b will parse strings using the actn.b file.

If your grammar is too big for yacc, you may try /crp/scj/bigyacc, some of whose size limits are larger, and others smaller.

FILES

actn.b output tables
actn.tmp temporary storage
Note that these files are created in the invoker's directory. The file actn.tmp is only created by /crp/scj/bigyacc (see above).

SEE ALSO

Yacc manual, by scj (available from ek); "LR Parsing", by A. V. Aho and S. C. Johnson, to be published.

DIAGNOSTICS

There are various diagnostics, but only one can be obtained in each run.

BUGS

The maximum number of terminal and non-terminal symbols is 50 each, and this is not checked. There are undoubtedly other bugs too.

11/3/71 FORM (I)

NAME form -- form letter generator

SYNOPSIS form proto arg_1 ...

DESCRIPTION form generates a form letter from a prototype letter, an associative memory, arguments and in a special case, the current date.

If form is invoked with the argument x, the following files come into play:

x.f prototype input
x.r form letter output
x.am associative memory
form.am associative memory if x.am not found.

Basically, form is a copy process from the file x.f to the file x.r. If an element of the form \n (where n is a digit from 1 to 9) is encountered, The nth argument is inserted in its place, and that argument is then rescanned. If \0 is encountered, the current date is inserted. If the desired argument has not been given, a message of the form "\n: " is typed. The response typed in then is used for that argument.

If an element of the form [name] is encountered, the name is looked up in the associative memory. If it is found, the contents of the memory under this name replaces the original element (again rescanned.) If the name is not found, a message of the form "name: " is typed. The response typed in is used for that element. If the associative memory is writable, the response is entered in the memory under the name. Thus the next search for that name will succeed without interaction.

In both of the above cases, the response is typed in by entering arbitrary text terminated by two new lines. Only the first of the two new lines is passed with the text. The process is instantly terminated if an end of file is encountered anywhere except in the associative memory.

FILES

x.f	input file
x.r	output file
x.am	associative memory
form.am	associative memory

SEE ALSO type

DIAGNOSTICS "settup error" when the appropriate files cannot be located or created.

BUGS "settup" is misspelled.

TYPO (I) 1/15/73 TYPO (I)

NAME typo -- find possible typo's

SYNOPSIS typo [-] file$_1$ ...

DESCRIPTION typo hunts through a document for unusual words, typographic errors, and hapax legomena and prints them on the standard output.

All words used in the document are printed out in decreasing order of peculiarity along with an index of peculiarity. An index of 10 or more is considered peculiar. Printing of certain very common English words is suppressed.

The statistics for judging words are taken from the document itself; with some help from known statistics of English. The "-" option suppresses the help from English and should be used if the document is written in, for example, Urdu.

Roff and Nroff control lines are ignored. Upper case is mapped into lower case. Quote marks, vertical bars, hyphens, and ampersands are stripped from within words. Words hyphenated across lines are put back together.

FILES /tmp/ttmp??, /etc/salt, /etc/w2006

SEE ALSO --

DIAGNOSTICS yes, lots

BUGS Because of the mapping into lower case and the stripping of special characters, words may be hard to locate in the original text.

NAME ov — overlay pages

SYNOPSIS ov [file]

DESCRIPTION ov is a postprocessor for producing double column formatted text when using nroff(I). ov literally overlays successive pairs of 66-line pages.

If the file argument is missing, the standard input is used. Thus ov may be used as a filter.

FILES none

SEE ALSO nroff(I), pr(I)

DIAGNOSTICS none

BUGS Other page lengths should be permitted.

NAME

wwb, style, diction, punct − writers workbench

SYNOPSIS

wwb style [option] ... file ...

wwb diction [option] ... file ...

wwb suggest

wwb punct file ...

wwb wwb [**−ms**] [**−mm**] file ...

DESCRIPTION

Wwb controls many subprograms documented in the references. For the full treatment use 'wwb wwb', but for quick use the following subprograms are recommended.

Style reports on readability, sentence length and structure, word length and usage, verb type, and sentence openers in the named documents.

Diction prints all sentences that contain phrases from a list of bad or verbose word patterns. Option **−p** *pfile* supplies an additional private pattern file; option **−s** skips the standard one.

Suggest reads, from the standard input, phrases deprecated by *diction* and proposes alternatives.

Punct reports sentences that appear to violate standard punctuation rules and sentences that contain doubled words.

Style and *diction* expect documents prepared for *nroff*/*troff*(1) and the preprocessors *pic*, *ideal*, *grap*, *eqn*, *refer*, and *tbl*(1). Option **−mm** overrides the default macro package, **−ms**. The related option **−ml** skips mm-style lists, and should be used if there are many lists of non-sentences.

Other options for *style* are:

−a print all sentences with their length and readability index.

−e print all sentences that begin with an expletive.

−p print all sentences that contain a passive verb.

−l*num* print all sentences longer than *num*.

−r*num* print all sentences whose readability index is greater than *num*.

−P print parts of speech of the words in the document.

SEE ALSO

deroff(1), spell(1)

Writer's Workbench User's Manual, /usr/man/manw/*

BUGS

Use of non-standard formatting macros may cause incorrect sentence breaks.
Imperatives cannot be recognized.

NAME

sky – obtain ephemerides

SYNOPSIS

sky

DESCRIPTION

Sky predicts the apparent locations of the Sun, the Moon, the planets out to Saturn, stars of magnitude at least 2.5, and certain other celestial objects including comet Kohoutek and M31. *Sky* reads the standard input to obtain a GMT time typed on one line with blanks separating year, month number, day, hour, and minute; if the year is missing the current year is used. If a blank line is typed the current time is used. The program prints the azimuth, elevation, and magnitude of objects which are above the horizon at the ephemeris location of Murray Hill at the indicated time.

Placing a "1" input after the minute entry causes the program to print out the Greenwich Sidereal Time at the indicated moment and to print for each body its right ascension and declination as well as its azimuth and elevation. Also, instead of the magnitude, the geocentric distance of the body, in units the program considers convenient, is printed. (For planets the unit is essentially A. U.)

The magnitudes of Solar System bodies are not calculated and are given as 0. The effects of atmospheric extinction are not included; the mean magnitudes of variable stars are marked with "v".

For all bodies, the program takes into account precession and nutation of the equinox, annual (but not diurnal) aberration, diurnal parallax, and the proper motion of stars (but not annual parallax). In no case is refraction included.

The program takes into account perturbations of the Earth due to the Moon, Venus, Mars, and Jupiter. The expected accuracies are: for the Sun and other stellar bodies a few tenths of seconds of arc; for the Moon (on which particular care is lavished) likewise a few tenths of seconds. For the Sun, Moon and stars the accuracy is sufficient to predict the circumstances of eclipses and occultations to within a few seconds of time. The planets may be off by several minutes of arc.

Information about the program may be obtained from its author.

FILES

/usr/lib/startab, /usr/lib/moontab

SEE ALSO

azel (VI)

American Ephemeris and Nautical Almanac, for the appropriate years; also, the *Explanatory Supplement to the American Ephemeris and Nautical Almanac.*

AUTHOR

R. Morris

NAME

wwv − print and set the date from accurate clock

SYNOPSIS

wwv [−b −s −f −u]

DESCRIPTION

wwv connects to a clock synchronized with radio station WWV. With no argument it prints the time obtained. It accepts these arguments:

−b Print both the WWV time and the system's current idea of the time.

−s Set the system's time to agree with WWV. You must be super-user. The system time cannot be changed by more than 20 minutes.

−f With −s, force the time to be set by WWV even if the local time disagrees by more than 20 minutes.

−u Report time in UTC rather than local time.

FILES

/usr/adm/wtmp to record time-setting

SEE ALSO

date (1), utmp(5)

BUGS

The WWV clock has been observed to slip sync by an integral number of minutes or hours from the radio standard.

NAME

cdl − circuit description language

DESCRIPTION

The circuit descriptions used by design aid programs *place*, *wcheck*, *bdraw* and *wrap* are expressed in the Circuit Design Language described below. A complete description has four main parts which must appear in the sequence: Package Descriptions, Board Description, Component Descriptions, then Circuit Description. The last of these can be generated automatically from circuit drawings made by the command *draw*.

The elements of a circuit are as follows.

signal	The information that passes over a wire and is carried by a net of wires.
chip	A circuit component.
package	The physical unit which houses one or more chips.
pin	The point where a signal wire connects to a chip.
board	The physical unit on which packages are mounted.
socket	The device for mounting a package on a board.
connector	The interface between signal wires on the board and those external to the board.

Chips, packages and sockets are grouped by type. All the items of one type exhibit the same properties. Types, like signals, chips and connectors, are identified by name. Pins and sockets are identified by number.

A name is a string of up to 16 letters, digits or any of the symbols +−.$ with the restriction that the first character cannot be a digit. In the simplest case, no two names can be the same but there is provision for dividing a circuit into *modules* so that a name in one module is not confused with the same name appearing in another module. A name that is defined prior to the first .m command is *global* and must be unique. (The description of a module starts with .m) A name defined within a module is a *local* name and must not be the same as either another name local to the same module or a global name. The full form of a local name consists of the module name and the local name separated by a /. It cannot be longer than 16 characters. Within the text of one module description it is possable to refer to an element of another module by giving its full name. A name prefixed with is a global name.

The circuit board and components mounted on it are described as rectangles. They are positioned so that their sides are parallel to one or other of the axes used to describe circuit board geometry. The axes should be chosen so that the following are true.

The origin of the coordinate system lies outside the board and the board is in the first quadrant.

When the board is loaded with dual-in-line packages, pin one of a package is the closest pin to the board origin.

The rows of pins on dual-in-line packages run parallel to the X axis.

Measurements are expressed in hundredths of an inch. All are integers and have no explicit decimal point. Coordinates are expressed as pairs of integers spearated by '/'. The X coordinate appears first.

It is sometimes necessary to provide a list of coordinates. Invariably each coordinate is associated with a numbered item (pin number, socket number etc.). A one item list consists of the item number followed by its coordinates as in

28 170/250

A series of equally spaced and consecutively numbered items can be described by giving the first and last item descriptions and separating the two with '-' as in

28 170/250 - 30 190/200

(item number 29 appears at position 180/225). If the item numbers are equally spaced but not

consecutive a step size can follow the '-' as in

12 200/700 -9 147 200/100

(which describes the positions of items numbered 12, 21, 30 etc.)

In the following list of commands, characters that must appear are written in **bold** type while character strings are identified by names in *italics*. Brackets [and] enclose an optional item and vertical bar | indicates an alternative. A list of items and coordinates, written as described in the previous paragraph, is represented by

setof{*item coordinate*}

Commands are separated by either newline or semi-colon. A comment can appear on any line. It starts with a % and ends with a newline.

Lines that do not start with a period are signal definition lines. They have the format

name [**pin**] ... | *name* [= *signal*]

The signal called *name* is described either as being connected to one or more *pins*, or is described as being equivalent to a previously described *signal*. Signal descriptions of this form can appear after .c commands, in which case the pin numbers refer to pins on a chip, or they appear after .i and .o commands, in which case the pins are those on a connector.

The commands are as follows.

Package Description

.k *name pins length width*
Define package type. Arguments are the package type *name*, the number of *pins* on each package, and its overall dimensions. The *length* is the distance along the X axis and the *width* is the distance along the Y axis. Note: the pins for a package should be numbered consecutively from one even if that means inventing a few extra pins.

.kp setof{*pinno coord*}
Package pin locations. One or more **.kp** commands following a **.k** command gives the list of pins and their coordinates relative to the origin of the package. (By convention, the origin of a dual-in-line package is that corner of the package closest to pin number one.)

Board Description

.b *name sockets connectors* Define a circuit board. Arguments are board *name*, the number of *sockets* on the board, and the number of I/O *connectors*.

.bs [+ | −] *rotation coord*
Board Orientation. A **.bs** command gives the orientation of the board relative to that used when the board is placed in a wire-wrap machine. A - indicates that the board description is with the pins pointing up. A + is used when the board is described from the component side. *Coord* is the board location of the wiring machine's origin and *rotation* is the number of right-angles by which the board must be rotated anti-clockwise about the wiring machine's origin in order to bring the board into the first quadrant.

.s *name* [*package*] ...
Define a socket type. Arguments are the socket type *name* and a list of the *package* types that can be located in the socket. The first *package* listed should be that whose pin numbering is the same as is used on the socket.

.sb setof{*socketno coord*}
Socket locations. One or more **.sb** commands following a **.s** command give the list of sockets mounted on the circuit board. Each socket has a number and board location. The socket numbers must run consecutively from one. The position of a socket is that of the corner nearest to the board origin. **.x** *name pins length width*
Define Input/Output Connector. Arguments are the *name* of the connector, the number of *pins*, and its overall dimensions. The *length* is the distance along the X axis and the *width* is the

distance along the Y axis. Note: the pins for a connector should be numbered consecutively from one even if that means inventing a few extra pins.

.xb *coord*
Connector location. The *coord* is the board position of the connector. That is the position of the corner nearest to the board origin.

xp setof{*pinno coord*}
Connector pin positions. One or more **.xp** commands following a **.x** command gives the list of pins and their coordinates relative to the position of the connector.

.v *number name*
Define special signal. The special signals are numbered consecutively from zero. The arguments for a **.v** are the special signal *number* and the signal *name* to which it corresponds.

.vb setof{*pinno coord*}
Special signal pin positions. One or more **.vb** commands following a **.v** command give the list of pins and their positions on the circuit board. The pins should be numbered consecutively from one.

Chip Type Description

.t *name package* [*pinno*] ... | **.t** *name* = *type*
Define a chip type. Arguments are the chip type *name*, the name of the *package* in which it is installed, and pin numbers, *pinno*, for the special voltage connections. The special voltage pin numbers can be omitted if none are to be wired but if any are present they must be in the same sequence with which the special signals are numbered. (See **.v** command above.) A **.t** command can be used to give an alternate name to a chip type. In that case the alternate *name* should be followed by equals and then the already defined *type* name.

.to [*pinno*] ...
Chip output pins. One or more **.to** commands following a **.t** command give the numbers of pins that carry output signals from the chip.

Circuit Description

name Start a module. The module
is given as argument. All unqualified chip and signal names (no '/') in subsequent text are presumed to belong to this module.

.p *pageno*
Start a page. The page number is *pageno*.

.c *name* [**type**] | **.c** *name* = *chip*
Describe a chip. If the chip has been previously described one can give just one argument, the chip *name*. Otherwise one must give the chip *name* and chip *type*, or the chip *name* and the name of a previously described *chip*. In the latter case the two chips will be treated as one. Signal descriptions following a **.c** command refer to pins on the chip.

.c[**h** | **s**] *socketno*
Chip placement. *Socketno* is the number of the socket into which the chip described in a previous **.c** command is to be placed. A hard placement, specified by **.ch**, will not be changed by automatic placement programs whereas a soft placement, specified by **.cs**, will. A placement specified by a **.cu** command is a dummy that is used to describe chip grouping only. In all cases, chips assigned to the same *socketno* occupy a single package.

.i *name*
Circuit board inputs. The connector *name* is given as argument. Signal descriptions following this command refer to signals originating outside the circuit board and the pin numbers are connector pin numbers.

.o *name*
Circuit board output. The connector *name* is given as argument. Signal descriptions following this command refer to signals originating on the circuit board and the pin numbers are connector pin numbers.

.h *signal*
Hand wired signal. The argument is the *name* of a signal that will be ignored by an automatic wiring program.

.u
Un-fix chip positions. Chip positions defined with previous **.ch**, **.cs** or **.cu** commands will have no effect on the placement. However, grouping of chips will remain unchanged.

.q
End of file. Not usually necessary.

SEE ALSO
draw(1), wcheck(1), wrap(1), place(1), bdraw(1)

NAME tss -- interface to Honeywell TSS

SYNOPSIS tss

DESCRIPTION tss will call the Honeywell 6070 on the 201 data phone. It will then go into direct access with TSS. Output generated by TSS is typed on the standard output and input requested by TSS is read from the standard input with UNIX typing conventions.

An interrupt signal (ASCII DEL) is transmitted as a "break" to TSS.

Input lines beginning with ! are interpreted as UNIX commands. Input lines beginning with ~ are interpreted as commands to the interface routine.

~<file insert input from named UNIX file

~>file deliver tss output to named UNIX file

~p pop the output file

~q disconnect from tss (quit)

~r file receive from HIS routine CSR/DACCOPY

~s file send file to HIS routine CSR/DACCOPY

Ascii files may be most efficiently transmitted using the HIS routine CSR/DACCOPY in this fashion. Underlined text comes from TSS. AFTname is the 6070 file to be dealt with.

SYSTEM? CSR/DACCOPY (s) AFTname
Send Encoded File ~s file

SYSTEM? CSR/DACCOPY (r) AFTname
Receive Encoded File ~r file

FILES /dev/dn0, /dev/dp0

SEE ALSO --

DIAGNOSTICS DONE when communication is broken.

BUGS When diagnostic problems occur, tss exits rather abruptly.

OWNER csr

NAME

nfs − communicate with Spider File Store

SYNOPSIS

nfs −key [name ...]

DESCRIPTION

Nfs saves and restores selected portions of the file system hierarchy in the Spider File Store(FS). Its actions are controlled by the *key* argument. The key is a string of characters preceded by '−' and containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be transfered, or listed. If no name is given, '.' is the default.

The name arguments serve a dual role when a file is being transfered between UNIX and FS. In UNIX they are interpreted as usual with respect to the current directory; in the File Store, they are interpreted with respect to the user's directory (as specified in the File Store password list). Thus, if the user 'abc' has a FS login directory abc, the file names in the argument refer to files in the FS directory abc regardless of the current UNIX directory. For example, if the user abc's current working directory is /usr/hem/second and he invokes

nfs −w x y

the files x and y of /usr/hem/second are sent to the FS files /abc/x and /abc/y.

If c is used the arguments are interpreted with respect to a named directory; if '.' is used the FS interprets the files names with respect to the current working UNIX directory. For example, if abc's working directory is /usr/abc/sub/dir and he invokes

nfs −w. x y

the files are put on FS /abc/sub/dir/x and /abc/sub/dir/y. Without the '.' option they are sent to the FS /abc/x and /abc/y. Normally, when a directory name is specified it denotes the contents of that directory. If x is used then the directory name denotes the directory itself.

Each time that *nfs* is used the user is logged into the File Store and for this purpose his current user name (login) under UNIX is used.

The function portion of the key is specified by one of the following letters:

w Write the indicated files and directories to the File Store. If files with the same names already exist, they are replaced.

u Update the File Store. **u** is the same as **w**, but a file is replaced only if its modification date is later than the date stored in the File Store; i.e. if it has changed since it was last written there.

d Delete the named files and directories from the File Store. At least one name argument must be given.

r Read the named files from the FS and write them into the UNIX file system.

t List the titles of all items in the named directories. The complete tree is given. Names begining with '.' are not listed. If a file name is specified that name is typed back but only if the file exists in the FS.

l List is the same as **t** but more information about each file and directory is given.

s The space allocation and amount of space used for the account associated with each named directory is given.

m Make a new directory in the FS.

The following characters may be used in addition to the letter which selects the function desired.

c — The current directory in the file store is specified in the first name argument.

. — The FS has the same working directory as the user has on UNIX.

v — Set verbose mode. Normally *nfs* does its work silently. The v option causes it to type the name of each file it treats preceded by the function letter.

i — Set interactive mode. In this mode *nfs* pauses before treating each file, types the indicative letter and the file name, and awaits the user's response. Response y means 'yes', so the file is treated. Null response means 'no', and the file does not take part in whatever is being done. Response x means 'exit'; the *nfs* command terminates immediately.

x — Set explicit mode. In this mode directory names denote the directories and not their contents (as is normal).

T — Temporary files are created. This option is used for files which are transient on the FS and do not need tape backup.

B — Large files have their space allocated in tracks.

FILES

/dev/tiu

see ALSO

ufs(1)

NAME

dcon, ndcon, rx, rogin, rsh − remote login and execution

SYNOPSIS

dcon [option] ... machine

ndcon machine

rx machine [command-list]

/usr/bin/m/machine [command-list]

/usr/inet/bin/rogin machine [**−l** username]

/usr/inet/bin/rsh machine [**−l** username] [command-list]

DESCRIPTION

Dcon logs in to the computer whose Datakit address is *machine*. It is much like *cu*(1), but the only local escapes are hang up '~.' and a shell escape '~!'.

Dcon normally tries to log in automatically, using the login id of the invoking user. To login explicitly, or to connect to machines that disallow such access, use option **−l**. Other options are:

−v Verbose. Give play-by-play while logging in.

−s Script. The *machine* argument names a file that guides login. The first line of the file is the machine name. Later lines are paired: a prompt word expected from the remote machine (including nonblank punctuation), with an input line to send upon receiving that prompt. All other words received while looking for prompts are ignored.

Ndcon logs in to a remote computer similarly to *dcon*, but with a direct *stream*(4)-to-stream connection. In particular *mux*(9) layers may be downloaded across it. The only local escape is the quit signal (control-\). Legitimate answers to the subsequent prompt 'dcon>>' are 'i' [sic] to send the quit signal to the remote machine, 'x' or '.' to exit *ndcon*, and '!command-list' to execute commands locally.

Rx invokes a shell on the designated *machine* and passes the *command-list* to that shell. The standard input and output of the remote process are the standard input and output of *rx*. The standard error file from the remote process is the same as the standard output. The current directory, permissions and shell variables of the remote shell are what the user would get by logging in directly. Unquoted shell metacharacters are interpreted locally; quoted ones are interpreted on the remote machine.

Rx with no *command-list* is equivalent to 'dcon machine'.

Directory */usr/bin/m* contains machine names as commands: '/usr/bin/m/machine' with no argument gets an *ndcon* connection; with arguments it does *rx*. If the directory is in the *sh*(1) search path, the names become commands for navigating the local cluster.

Rogin and *rsh* are to ARPA internet as *dcon* and *rx* are to Datakit. A file '.rhosts' in the login directory for *username* on a receiving machine lists machine/user pairs that may log in as *username* without a password check. Pairs appear one per line separated by blanks.

EXAMPLES

rx overthere cat file1 > file2
: copies remote *file1* to local *file2*; for other ways to do the job, see *push*(1) and *nfs*(5)

rx overthere cat file1 ">" file2
: copies remote *file1* to remote *file2*

FILES

/usr/inet/lib/*
/usr/inet/lib/hosts.equiv list of machines with identical users
$HOME/.rhosts

NAME su -- become privileged user

SYNOPSIS su password

DESCRIPTION su allows one to become the super-user, who has all sorts of marvelous powers. In order for su to do its magic, the user must pass as an argument a password. If the password is correct, su will execute the shell with the UID set to that of the super-user. To restore normal UID privileges, type an end-of-file to the super-user shell.

FILES --

SEE ALSO shell

DIAGNOSTICS "Sorry" if password is wrong

BUGS --

OWNER dmr, ken

NAME

su – substitute user id temporarily

SYNOPSIS

/etc/su [userid]

DESCRIPTION

Su demands the password of the specified *userid*, and if it is given, changes to that *userid* and invokes the Shell *sh*(1) without changing the current directory. The user environment is unchanged except for HOME and SHELL, which are taken from the password file for the user being substituted (see *environ*(5)). The new user ID stays in force until the Shell exits.

If no *userid* is specified, 'root' is assumed. To remind the super-user of his responsibilities, the Shell substitutes '#' for its usual prompt.

SEE ALSO

sh(1)

CRYPT (I) 10/23/71 CRYPT (I)

NAME crypt — encode/decode

SYNOPSIS crypt [password]

DESCRIPTION crypt is an exact implementation of Boris Hagelin's cryptographic machine called the M-209 by the U. S. Army [1].

crypt reads from the standard input file and writes on the standard output. For a given password, the encryption process is idempotent; that is,

```
crypt znorkle <clear >cypher
crypt znorkle <cypher
```

will print the clear.

crypt is suitable for use as a filter:

```
pr <"crypt bandersnatch"<cypher
```

FILES —

SEE ALSO [1] U. S. Patent 2,089,603.

DIAGNOSTICS —

BUGS —

NAME

crypt, encrypt, decrypt – encode/decode

SYNOPSIS

/usr/games/crypt [*password*]

/usr/games/encrypt [**-p**] [*password*]
/usr/games/decrypt [**-p**] [*password*]

DESCRIPTION

These commands read from the standard input and write on the standard output. The *password* is an enciphering key. If no password is given, one is demanded from the terminal; echoing is turned off while it is being typed in. *Crypt* uses a relatively simple, fast method (rotor machine) for both enciphering and deciphering. *Encrypt* and *decrypt* use a more robust, slower method (DES). Files enciphered by *crypt* are compatible with those treated by the editor *ed* in encryption mode. Files enciphered by *crypt* are not intelligible to *encrypt*/*decrypt*, and vice versa.

It is prudent to supply the key from the terminal, not from the command line, and to pick a reasonably obscure and long key (6 letters for *crypt* and much longer for *encrypt*).

Under option **-p** *encrypt* enciphers into printing characters, which can be sent by *mail*(1). *Decrypt* can distinguish ciphertext from clear: it will work on a full mail message, headers and all.

FILES

/dev/tty for typed key

SEE ALSO

ed(1), *makekey*(8)
J. A. Reeds and P. J. Weinberger, 'File Security and the Unix Crypt Command,' *AT&T Bell Laboratories Technical Journal*, 63 (1984) 1673-1684

BUGS

Encipherment cannot frustrate adversaries with super-user privileges. Cryptogames have other dangers too. The only useful application is in data transmission.

NAME

number – convert Arabic numerals to English

SYNOPSIS

number

DESCRIPTION

Number copies the standard input to the standard output, changing each decimal number to a fully spelled out version. Punctuation is added to make the output sound well when played through *speak*(1).

SEE ALSO

speak(1)

11/3/71 DSW (I)

NAME dsw -- delete interactively

SYNOPSIS dsw [directory]

DESCRIPTION For each file in the given directory ("." if not specified) dsw types its name. If "y" is typed, the file is deleted; if "x", dsw exits; if anything else, the file is not removed.

FILES --

SEE ALSO rm

DIAGNOSTICS "?"

BUGS The name "dsw" is a carryover from the ancient past. Its etymology is amusing but the name is nonetheless ill-advised.

OWNER dmr, ken

NAME

dd – convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

Dd copies its specified input file to its specified output with possible conversions. The input and output block size may be specified to take advantage of raw physical I/O.

option	*values*
if=	input file name; standard input is default
of=	output file name; standard output is default
ibs=	input block size (default 512)
obs=	output block size (default 512)
bs=	set both input and output block size, superseding *ibs* and *obs;* also, if no conversion is specified, it is particularly efficient since no copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input records before starting copy
count=*n*	copy only *n* input records
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input record to *ibs*
... , ...	several comma-separated conversions

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b** or **w** to specify multiplication by 1024, 512, or 2 respectively. Also a pair of numbers may be separated by **x** to indicate a product.

Cbs is used only if *ascii* or *ebcdic* conversion is specified. In the former case *cbs* characters are placed into the conversion buffer, converted to ASCII, and trailing blanks trimmed and new-line added before sending the line to the output. In the latter case ASCII characters are read into the conversion buffer, converted to EBCDIC, and blanks added to make up an output record of size *cbs.*

After completion, *dd* reports the number of whole and partial input and output blocks.

For example, to read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into the ASCII file *x:*

dd if=/dev/rmt0 of=x ibs=800 cbs=80 conv=ascii,lcase

Note the use of raw magtape. *Dd* is quite suited to I/O on the raw physical devices because it allows reading and writing in arbitrary record sizes.

SEE ALSO

cp (I)

BUGS

The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. It is not clear how this relates to real life.

NAME

faced – network face server

SYNOPSIS

`/usr/net/face.go`

DESCRIPTION

The network face server provides a database of 48×48 bit icons and other facial representations. It is implemented as a network file system similar to *netfs*(8).

The file system, conventionally mounted on `/n/face`, has a fixed three-level hierarchy. The first level is a machine name, the second level a user name, and the third level a resolution. Thus the file `/n/face/kwee/pjw/48x48x1` is the standard face icon (for user pjw) on machine kwee:

Many local users also have 512×512 byte high-resolution faces, named `512x512x8`. Other resolutions may also be present for a particular face. One-bit images are stored in the format used by *icon*(9.1); eight-bit images are arrays of bytes. The directories for machines sharing a user community, such as those on a Datakit node, are linked together and given a name appropriate to the community. For example, `/n/face/kwee` is a link to `/n/face/astro`.

To access the face for a mail name *machine*`!`*uid* take the result of the first successful open from the following list of files:

```
/n/face/machine/uid/48x48x1
/n/face/misc./uid/48x48x1
/n/face/machine/unknown/48x48x1
/n/face/misc./unknown/48x48x1
```

The directory `misc.` holds faces for generic users such as `root` and `uucp`. The face server is made available on a machine by running `/usr/net/face.go` from *rc*(8).

The face server data is administered by a pair of ASCII files that associate related machines and faces. The machine table `machine.tab` attaches machines to communities; in it the line

```
kwee=astro
```

puts machine `kwee` in community `astro`. The people table `people.tab` associates a machine/user pair in the face server with a file in one of the source directories `/usr/jerq/icon/face48` or `/t0/face/512x512x8` on kwee. Thus

```
astro/pjw=pjweinberger
```

causes the images stored in source files named `pjweinberger` to be available in the face server in directory `/n/face/astro/pjw`. As well, each disk file used by the face server is linked (by its original name) into the directory `/n/face/48x48x1` or `/n/face/512x512x8` for easy access to all the images.

FILES

`/n/kwee/usr/jerq/icon/face48`	directory of low resolution faces
`/n/kwee/t0/face/512x512x8`	directory of high resolution faces
`/n/kwee/usr/net/face/people.tab`	people/file equivalences
`/n/kwee/usr/net/face/machine.tab`	machine/community equivalences

SEE ALSO

netfs(8), *face*(9.1), *mugs*(9.1), *icon*(9.1), *vismon*(9.1)

BUGS

After updating the tables, an indeterminate time may pass before the new faces are available.
All face server files are unwritable.

OSSANNA

UNIX PROGRAMMER'S MANUAL

K. Thompson

D. M. Ritchie

November 3, 1971

INTRODUCTION

This manual gives complete descriptions of all the publicly available features of UNIX. It provides neither a general overview (see "The UNIX Time-sharing System" for that) nor details of the implementation of the system (which remain to be disclosed).

Within the area it surveys, this manual attempts to be as complete and timely as possible. A conscious decision was made to describe each program in exactly the state it was in at the time its manual section was prepared. In particular, the desire to describe something as it should be, not as it is, was resisted. Inevitably, this means that many sections will soon be out of date. (The rate of change of the system is so great that a dismayingly large number of early sections had to be modified while the rest were being written. The unbounded effort required to stay up-to-date is best indicated by the fact that several of the programs described were written specifically to aid in preparation of this manual!)

This manual is divided into seven sections:

I. Commands
II. System calls
III. Subroutines
IV. Special files
V. File formats
VI. User-maintained programs
VII. Miscellaneous

Commands are programs intended to be invoked directly by the user, in contradistinction to subroutines, which are intended to be called by the user's programs. Commands generally reside in directory /bin (for binary programs). This directory is searched automatically by the command line interpreter. Some programs classified as commands are located elsewhere; this fact is indicated in the appropriate sections.

System calls are entries into the UNIX supervisor. In assembly language, they are coded with the use of the opcode "sys", a synonym for the trap instruction.

The special files section discusses the characteristics of each system "file" which actually refers to an I/O device.

The file formats section documents the structure of particular kinds of files; for example, the form of the output of the loader and assembler is given. Excluded are files used by only one command, for example the assembler's intermediate files.

User-maintained programs are not considered part of the UNIX system, and the principal reason for listing them is to indicate their existence without necessarily giving a complete description. The author should be consulted for information.

The miscellaneous section gathers odds and ends.

Each section consists of a number of independent entries of a page or so each. The name of the entry is in the upper right corner of its pages, its preparation date in the upper left. Entries within each section are alphabetized. It was thought better to avoid page numbers, since it is hoped that the manual will be updated frequently.

All entries have a common format.

The name section repeats the entry name and gives a very short description of its purpose.

The synopsis summarizes the use of the program being described. A few conventions are used, particularly in the Commands section:

Underlined words are considered literals, and are typed just as they appear.

Square brackets ([]) around an argument indicate that the argument is optional. When an argument is given as "name", it always refers to a file name.

Ellipses "..." are used to show that the previous argument-prototype may be repeated.

A final convertion is used by the commands themselves. An argument beginning with a minus sign "-" is often taken to mean some sort of flag argument even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with "-".

The description section discusses in detail the subject at hand.

The files section gives the names of files which are built into the program.

A see also section gives pointers to related information.

A diagnostics section discusses the diagnostics that may be produced. This section tends to be as terse as the diagnostics themselves.

The bugs section gives known bugs and sometimes deficiencies. Occasionally also the suggested fix is described.

The owner section gives the name of the person or persons to be consulted in case of difficulty. The rule has been that the last one to modify something owns it, so the owner is not necessarily the author. The owner's initials stand for

V7 V1

ken	K. Thompson
dmr	D. M. Ritchie
jfo	J. F. Ossanna
rhm	R. Morris

These three-character names also happen to be UNIX user ID's, so messages may be transmitted by the **mail** command or, if the addressee is logged in, by **write**.

At the beginning of this document is a table of contents, organized by section and alphabetically within each section. There is also a permuted index derived from the table of contents. Within each index entry, the title of the writeup to which it refers is followed by the appropriate section number in parentheses. This fact is important because there is considerable name duplication among the sections, arising principally from commands which exist only to exercise a particular system call.

This manual was prepared using the UNIX text editor **ed** and the formatting program **roff**.

1st edition

TABLE OF CONTENTS

No. 3171

I. COMMANDS

78 v1

II. SYSTEM CALLS

III. SUBROUTINES

IV. SPECIAL FILES

V. FILE FORMATS

VI. USER MAINTAINED PROGRAMS

VII. MISCELLANEOUS

UNIX PROGRAMMER'S MANUAL

Second Edition

K. Thompson

D. M. Ritchie

June 12, 1972

June 12, 1972

PREFACE
to the Second Edition

In the months since this manual first appeared, many changes have occurred both in the system itself and in the way it is used.

Perhaps most obviously, there have been additions, deletions, and modifications to the system and its software. It is these changes, of course, that caused the appearance of this revised manual.

Second, the number of people spending an appreciable amount of time writing UNIX software has increased. Credit is due to L. L. Cherry, M. D. McIlroy, L. E. McMahon, R. Morris, and J. F. Ossanna for their contributions.

Finally, the number of UNIX installations has grown to 10, with more expected. None of these has exactly the same complement of hardware or software. Therefore, at any particular installation, it is quite possible that this manual will give inappropriate information. One area to watch concerns commands which deal with special files (I/O devices). Another is places which talk about such things as absolute core locations which are likely to vary with the memory configuration and existence of protection hardware. Also, not all installations have the latest versions of all the software. In particular, the assembler and loader have just undergone major reorganizations in anticipation of a UNIX for the PDP-11/45.

81 v2

UNIX PROGRAMMER'S MANUAL

Third Edition

K. Thompson

D. M. Ritchie

February, 1973

February 1973

PREFACE
to the Third Edition

In the months since the last appearance of this manual, many changes have occurred both in the system itself and in the way it is used.

Perhaps most obviously, there have been additions, deletions, and modifications to the system and its software. It is these changes, of course, that caused the appearance of this revised manual.

Second, the number of people spending an appreciable amount of time writing UNIX software has increased. Credit is due to L. L. Cherry, M. D. McIlroy, L. E. McMahon, R. Morris, J. F. Ossanna, and E. N. Pinson for their contributions.

Finally, the number of UNIX installations has grown to 16, with more expected. None of these has exactly the same complement of hardware or software. Therefore, at any particular installation, it is quite possible that this manual will give inappropriate information.

In particular, any system which uses a PDP-11/20 processor will not include all the software described herein, nor will the software behave the same way. The second, or even the first, edition of this manual is likely to be more appropriate.

Besides additions, deletions, and modifications to the writeups in each section, this manual differs from its predecessors in two ways: all the commands used for system maintenance and not intended for normal users have been moved to a new section VIII; and there is a new "How to Get Started" chapter that gives some elementary facts and many pointers to other sections.

82 v3

INTRODUCTION TO THIS MANUAL

This manual gives descriptions of the publicly available features of UNIX. It provides neither a general overview (see "The UNIX Time-sharing System" for that) nor details of the implementation of the system (which remain to be disclosed).

Within the area it surveys, this manual attempts to be as complete and timely as possible. A conscious decision was made to describe each program in exactly the state it was in at the time its manual section was prepared. In particular, the desire to describe something as it should be, not as it is, was resisted. Inevitably, this means that many sections will soon be out of date. (The rate of change of the system is so great that a dismayingly large number of early sections had to be modified while the rest were being written. The unbounded effort required to stay up-to-date is best indicated by the fact that several of the programs described were written specifically to aid in preparation of this manual!)

This manual is divided into eight sections:

I. Commands
II. System calls
III. Subroutines
IV. Special files
V. File formats
VI. User-maintained programs
VII. Miscellaneous
VIII. Maintenance

Commands are programs intended to be invoked directly by the user, in contradistinction to subroutines, which are intended to be called by the user's programs. Commands generally reside in directory /bin (for binary programs). This directory is searched automatically by the command line interpreter. Some programs classified as commands are located elsewhere; this fact is indicated in the appropriate sections.

System calls are entries into the UNIX supervisor. In assembly language, they are coded with the use of the opcode "sys", a synonym for the trap instruction.

A small assortment of subroutines is available; they are described in section III. The binary form of most of them is kept in the system library /lib/liba.a.

The special files section IV discusses the characteristics of each system "file" which actually refers to an I/O device. Unlike previous editions, the names in this section refer to the DEC device names for the hardware, instead of the names of the special files themselves.

The file formats section V documents the structure of particular kinds of files; for example, the form of the output of the loader and assembler is given. Excluded are files used by only one command, for example the assembler's intermediate files.

User-maintained programs (section VI) are not considered part of the UNIX system, and the principal reason for listing them is to indicate their existence without necessarily giving a complete description. The author should be consulted for information.

The miscellaneous section (VII) gathers odds and ends.

Section VIII discusses commands which are not intended for use by the ordinary user, in some cases because they disclose information in which he is presumably not interested, and in others because they perform privileged functions.

Each section consists of a number of independent entries of a page or so each. The name of the entry is in the upper corners of its pages, its preparation date in the upper middle. Entries within each section are alphabetized. The page numbers of each entry start at 1. (The earlier hope for frequent, partial updates of the manual is clearly in vain, but in any event it is not feasible to maintain consecutive page numbering in a document like this.)

All entries have a common format.

The name section repeats the entry name and gives a very short description of its purpose.

The synopsis summarizes the use of the program being described. A few conventions are used, particularly in the Commands section:

Underlined words are considered literals, and are typed just as they appear.

Square brackets ([]) around an argument indicate that the argument is optional. When an argument is given as "name", it always refers to a file name.

Ellipses "..." are used to show that the previous argument-prototype may be repeated.

A final convention is used by the commands themselves. An argument beginning with a minus sign "-" is often taken to mean some sort of flag argument even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with "-".

The description section discusses in detail the subject at hand.

The files section gives the names of files which are built

into the program.

A see also section gives pointers to related information.

A diagnostics section discusses the diagnostics that may be produced. This section tends to be as terse as the diagnostics themselves.

The bugs section gives known bugs and sometimes deficiencies. Occasionally also the suggested fix is described.

Previous edition of this manual had an owner section, which has been dropped from this edition because the "owners" of many routines became fairly hard to pin down. The major contributors to UNIX, (cast in order of appearance) together with their login names and most notable contributions, are

ken	K. Thompson	(UNIX, many commands)
dmr	D. M. Ritchie	(many commands, as, ld, C)
jfo	J. F. Ossanna	(roff, nroff)
doug	M. D. McIlroy	(tmg, m6)
rhm	R. Morris	(dc, much of library)
lem	L. E. McMahon	(cref)
llc	L. L. Cherry	(form, fed, salloc)
csr	C. S. Roberts	(tss)
enp	E. N. Pinson	(proof)

At the beginning of this document is a table of contents, organized by section and alphabetically within each section. There is also a permuted index derived from the table of contents. Within each index entry, the title of the writeup to which it refers is followed by the appropriate section number in parentheses. This fact is important because there is considerable name duplication among the sections, arising principally from commands which exist only to exercise a particular system call.

This manual was prepared using the UNIX text editor ed and the formatting program roff.

The assistance of R. Morris is gratefully acknowledged.

HOW TO GET STARTED

This section provides the basic information you need to get started on UNIX: how to log in and log out, how to communicate through your terminal, and how to run a program.

Logging in

You must call UNIX from an appropriate terminal. UNIX supports ASCII terminals typified by the TTY 37, the GE Terminet 300, the Memorex 1240, and various graphical terminals on the one hand, and IBM 2741-type terminals on the other.

To use UNIX, you must have a valid UNIX user name, which may be obtained, together with the telephone number, from the system administrators.

The same telephone number serves terminals operating at all the standard speeds. After a data connection is established, the login procedure depends on what kind of terminal you are using.

TTY 37 terminal

UNIX will type out "login: "; you respond with your user name. From the TTY 37 terminal, and any other which has the "new-line" function (combined carriage return and linefeed), terminate each line you type with the "new line" key (not the "return" key).

300-baud terminals

Such terminals include the GE Terminet 300, most display terminals, Execuport, TI, and certain Anderson-Jacobson terminals. These terminals generally have a speed switch which should be set at "300" (or "30" for 30 characters per second) and a half/full duplex switch which should be set at full-duplex. (Note that this switch will often have to be changed since MH-TSS requires half-duplex). When a connection with UNIX is established, a few garbage characters are typed (the login message at the wrong speed). Depress the "break" key; this is a speed-independent signal to UNIX that a 300-baud terminal is in use. UNIX will type "login: " at the correct speed; you type your user name, followed by the "return" key. Henceforth, the "return", "new line", or "linefeed" keys will give exactly the same results. Each line must be terminated with one of these keys; no one is listening to you until the return is received.

Selectric terminals

From an IBM 2741 or the Anderson-Jacobson Selectric terminal, no message will appear. After the data connection is established, press the "return" key. UNIX should type "login: " as described above. If the greeting does not

appear after a few seconds, unlock the keyboard by switching the terminal to local and back to remote, and type "return". If necessary, hang up and try again; something has gone wrong.

For all these terminals, it is important that you type your name in lower case if possible; if you type upper case letters, UNIX will assume that your terminal cannot generate lower-case letters and will translate all subsequent upper-case letters to lower case.

The evidence that you have successfully logged in is that a UNIX program, the Shell, will type a "%" to you. (The Shell is described below under "How to run a program".

For more information, consult getty(VII), which discusses the login sequence in more detail, and dc(IV), which discusses typewriter I/O.

Logging out

There are three ways to log out:

You can simply hang up the phone. Hanging up is safe if you are at command level, that is, if the Shell has just typed its prompt signal "%". It is also safe if you are in interactive system programs, for example the editor. It is unsafe if you are executing a non-interactive program, or one of your own programs, which either does not read the typewriter or ignores the end-of-file indications which will result from hanging up. The reason is that UNIX, unlike most systems, does not terminate a program simply because it has been hung-up upon.

You can log out by typing an end-of-file indication (EOT character, control "d") to the Shell. The Shell will terminate and the "login:" message will appear again.

You can also log in directly as another user by giving a login command (login (I)).

How to communicate through your terminal

When you type to UNIX, a gnome deep in the system is gathering your characters and saving them in a secret place. The characters will not be given to a program until you type a return, as described above in Logging in.

UNIX typewriter I/O is full-duplex (except for Selectric terminals). It has full read-ahead, which means that you can type at any time, even while a program is typing at you. Of course, if you type during output, the output will have the input characters interspersed. However, whatever you type will be saved up and interpreted in correct sequence.

There is a limit to the amount of read-ahead, but it is generous

and not likely to be exceeded unless the system is in trouble. When the read-ahead limit is exceeded, the system stops echoing input characters, and starts echoing "#" no matter what you typed. The last character which was echoed correctly will be received correctly by the program to which you were talking; subsequent characters have been thrown away.

On a typewriter input line, the character "@" kills all the characters typed before it, so typing mistakes can be repaired on a single line. Also, the character "#" erases the last character typed. Successive uses of "#" erase characters back to, but not beyond, the beginning of the line. "@" and "#" can be transmitted to a program by preceding them with "\". (So, to erase "\", you need two "#"s).

The ASCII "delete" (a.k.a. "rubout") character is not passed to programs but instead generates an interrupt signal. This signal generally causes whatever program you are running to terminate. It is typically used to stop a long printout that you don't want. However, programs can arrange either to ignore this signal altogether, or to be notified when it happens (instead of being terminated). The editor, for example, catches interrupts and stops what it is doing, instead of terminating, so that an interrupt can be used to halt an editor printout without losing the file being edited.

The quit signal is generated by typing the ASCII FS character. It not only causes a running program to terminate but also generates a file with the core image of the terminated process. Quit is useful for debugging.

Besides adapting to the speed of the terminal, UNIX tries to be intelligent about whether you have a terminal with the "new line" function or whether it must be simulated with carriage-return and line-feed. In the latter case, all input carriage returns are turned to new-line characters (the standard line delimiter) and both a carriage return and a line feed are echoed to the terminal. If you get into the wrong mode, the stty command (I) will rescue you.

Tab characters are used freely in UNIX source programs. If your terminal does not have the tab function, you can arrange to have them turned into spaces during output, and echoed as spaces during input. The system assumes that tabs are set every eight columns. Again, the stty command (I) will set or reset this mode. Also, there is a file which, if printed on TTY 37 or TerminNet 300 terminals, will set the tab stops correctly (tabs(VII)).

Section dc(IV) discusses typewriter I/O more fully. Section kl(IV) discusses the console typewriter.

How to run a program; The Shell

When you have successfully logged into UNIX, a program called the Shell is listening to your terminal. The Shell reads typed-in

lines, splits them up into a command name and arguments, and executes the command. A command is simply an executable program. The Shell looks first in your current directory (see next section) for a program with the given name, and if none is there, then in a system directory. There is nothing special about system-provided commands except that they are kept in a directory where the Shell can find them.

The command name is always the first word on an input line; it and its arguments are separated from one another by spaces.

When a program terminates, the Shell will ordinarily regain control and type a "%" at you to indicate that it is ready for another command.

The Shell has many other capabilities, which are described in detail in section sh(I).

The current directory

UNIX has a file system arranged in a hierarchy of directories. When the system administrator gave you a user name, he also created a directory for you (ordinarily with the same name as your user name). When you log in, any file name you type is by default in this directory. Since you are the owner of this directory, you have full permissions to read, write, alter, or destroy its contents. Permissions to have your will with other directories and files will have been granted or denied to you by their owners. As a matter of observed fact, few UNIX users protect their files from destruction, let alone perusal, by other users.

To change the current directory (but not the set of permissions you were endowed with at login) use chdir(I).

Path names

To reference files not in the current directory, you must use a path name.

Full path names begin with "/", the name of the root directory of the whole file system. After the slash comes the name of each directory containing the next sub-directory (followed by a "/"), until finally the file name is reached. E.g.: "/usr/lem/filex" refers to file "filex" in directory "lem"; "lem" is itself a sub-directory of "usr"; "usr" springs directly from the root directory.

If your current directory has subdirectories, the path names of files therein begin with the name of the subdirectory (no prefixed "/").

Without important exception, a path name may be used anywhere a file name is required.

Important commands which modify the contents of files are cp(I),

mv(I), and rm(I), which respectively copy, move (i.e. rename) and remove files. To find out the status of files or directories, use ls(I) and stat(I). See mkdir(I) for making directories; rmdir(I) for destroying them.

For a fuller discussion of the file system, see MM-71-1273-4. It may also be useful to glance through section II of this manual, which discusses system calls, even if you don't intend to deal with the system at the assembly-language level.

Writing a program

To enter the text of a source program into a UNIX file, use ed(I). The three principal languages in UNIX are assembly language (see as(I)), Fortran (see fc(I)), and C (see cc(I)). After the program text has been entered through the editor and written on a file, you can give the file to the appropriate language processor as an argument. The output of the language processor will be left on a file in the current directory named "a.out". (If the output is precious, use mv to move it to a less exposed name soon.) If you wrote in assembly language, you will probably need to load the program with library subroutines; see ld(I). The other two language processors call the loader automatically.

When you have finally gone through this entire process without provoking any diagnostics, the resulting program can be run by giving its name to the Shell in response to the "%" prompt.

The next command you will need is db(I). As a debugger, db is better than average for assembly-language programs, marginally useful for C programs (when completed, cdb(I) will be a boon), and virtually useless for Fortran.

Your programs can receive arguments from the command line just as system programs do. For assembly language programs, see exec(II).

Text processing

Almost all text is entered through the editor. The commands most often used to write text on a terminal are: cat(I), pr(I), roff(I), or nroff(I).

The cat command simply dumps ASCII text on the terminal, with no processing at all. The pr command paginates the text and supplies headings. The nroff command is an elaborate text formatting program, and requires careful forethought in entering both the text and the formatting commands into the input file. The roff command is a somewhat less elaborate formatting program, and requires somewhat less forethought.

Surprises

Certain commands provide inter-user communication. Even if you do not plan to use them, it would be well to learn something

about them, because someone else may aim them at you.

To communicate with another user currently logged in, write(I) is used. To leave a message the presence of which will be announced to another user when he next logs in, mail(I) is used. The write-ups in the manual also suggest how to respond to the two commands if you are a target.

When you log in, a message-of-the-day may greet you before the first "%".

TABLE OF CONTENTS

I. COMMANDS

UNIX PROGRAMMER'S MANUAL

Fourth Edition

K. Thompson

D. M. Ritchie

November, 1973

Nov 1973

PREFACE
to the Fourth Edition

In the months since the last appearance of this manual, many changes have occurred both in the system itself and in the way it is used. The most important changes result from a complete rewrite of the UNIX system in the C language. There have also been substantial changes in much of the system software. It is these changes, of course, which mandated the new edition of this manual.

The number of UNIX installations is now above 20, and many more are expected. None of these has exactly the same complement of hardware or software. Therefore, at any particular installation, it is quite possible that this manual will give inappropriate information. In particular, *the information in this manual applies only to UNIX systems which operate under the C language versions of the system.* Installations which use older versions of UNIX will find earlier editions of this manual more appropriate to their situation.

Even in installations which have the latest versions of the operating system, not all the software and other facilities mentioned herein will be available. For example, the typesetter, voice response unit, and voice synthesizer are hardly universally available devices, also, some of the UNIX software has not been released for use outside the Bell System.

The authors are grateful to L. L. Cherry, M. E. Lesk, E. N. Pinson, and C. S. Roberts for their contributions to the system software, and to L. E. McMahon for software and for his contributions to this manual. We are particularly appreciative of the invaluable technical, editorial, and administrative efforts of J. F. Ossanna, M. D. McIlroy, and R. Morris. They all contributed greatly to the stock of UNIX software and to this manual. Their inventiveness, thoughtful criticism, and ungrudging support increased immeasurably not only whatever success the UNIX system enjoys, but also our own enjoyment in its creation.

88 V4

TABLE OF CONTENTS

I. COMMANDS

II. SYSTEM CALLS

UNIX PROGRAMMER'S MANUAL

Fifth Edition

K. Thompson

D. M. Ritchie

June, 1974

June 1974

PREFACE
to the Fifth Edition

The number of UNIX installations is now above 50, and many more are expected. None of these has exactly the same complement of hardware or software. Therefore, at any particular installation, it is quite possible that this manual will give inappropriate information.

The authors are grateful to L. L. Cherry, L. A. Dimino, R. C. Haight, S. C. Johnson, B. W. Kernighan, M. E. Lesk, and E. N. Pinson for their contributions to the system software, and to L. E. McMahon for software and for his contributions to this manual. We are particularly appreciative of the invaluable technical, editorial, and administrative efforts of J. F. Ossanna, M. D. McIlroy, and R. Morris. They all contributed greatly to the stock of UNIX software and to this manual. Their inventiveness, thoughtful criticism, and ungrudging support increased immeasurably not only whatever success the UNIX system enjoys, but also our own enjoyment in its creation.

CA 90

UNIX PROGRAMMER'S MANUAL

Sixth Edition

K. Thompson

D. M. Ritchie

May, 1975

May 1975

PREFACE
to the Sixth Edition

We are grateful to L. L. Cherry, R. C. Haight, S. C. Johnson, B. W. Kernighan, M. E. Lesk, and E. N. Pinson for their contributions to the system software, and to L. E. McMahon for software and for his contributions to this manual. We are particularly appreciative of the invaluable technical, editorial, and administrative efforts of J. F. Ossanna, M. D. McIlroy, and R. Morris. They all contributed greatly to the stock of UNIX software and to this manual. Their inventiveness, thoughtful criticism, and ungrudging support increased immeasurably not only whatever success the UNIX system enjoys, but also our own enjoyment in its creation.

UNIX™ TIME-SHARING SYSTEM:

UNIX PROGRAMMER'S MANUAL

Seventh Edition, Volume 1

January, 1979

Bell Telephone Laboratories, Incorporated
Murray Hill, New Jersey

PREFACE

Although this Seventh Edition no longer bears their byline, Ken Thompson and Dennis Ritchie remain the fathers and preceptors of the UNIX† time-sharing system. Many of the improvements here described bear their mark. Among many, many other people who have contributed to the further flowering of UNIX, we wish especially to acknowledge the contributions of A. V. Aho, S. R. Bourne, L. L. Cherry, G. L. Chesson, S. I. Feldman, C. B. Haley, R. C. Haight, S. C. Johnson, M. E. Lesk, T. L. Lyon, L. E. McMahon, R. Morris, R. Muha, D. A. Nowitz, L. Wehr, and P. J. Weinberger. We appreciate also the effective advice and criticism of T. A. Dolotta, A. G. Fraser, J. F. Maranzano, and J. R. Mashey, and we remember the important work of the late Joseph F. Ossanna.

B. W. Kernighan
M. D. McIlroy

92 v7

†UNIX is a Trademark of Bell Laboratories

UNIX™ TIME-SHARING SYSTEM

PROGRAMMER'S MANUAL

Research Version

Eighth Edition, Volume 1

February, 1985

AT&T Bell Laboratories
Murray Hill, New Jersey

TABLE OF CONTENTS

GLOSSARY

This glossary covers major terms that have special meaning for the UNIX system. It excludes ordinary terms of art such as 'ASCII', 'compiler', 'address space', or 'byte'. It also excludes most terms peculiar to a single part of UNIX, e.g. 'diversion' (troff), 'enumeration' (C), or 'pattern space' (sed).

a.out the default name of a freshly compiled *object file*, pronounced 'A-dot-out'; historically a.out signified assembler output.

absolute pathname same as *full pathname*.

alarm a *signal* scheduled by the clock.

archive 1. a collection of data gathered from several *files* into one file. 2. especially, such a collection gathered by *ar*(1) for use as a *library*.

argument 1. a string made available to a *process* upon *executing* a *file*. 2. a string in a *command*, which the *shell* will pass to the command program as an argument [1].

ASCII file same as *text file*.

automatic persistent only during the invocation of a procedure, said of data belonging to a *process;* automatic data occupies the *stack segment;* cf. *static*.

background running independently of a terminal, said of a *process;* converse of *foreground*.

blit vernacular name for a prototype Teletype 5620 terminal; cf. *jerq*.

block the basic unit of *buffering* in the *kernel*, 1024 or 4096 bytes in the 8th edition.

block device a *device* upon which a *file system* [1] can be *mounted*, typically a permanent storage device such as a tape or disk drive, so called because data transfers to the device occur by *blocks;* cf. *character device*.

boot to start the operating system, so called because the *kernel* must bootstrap itself from secondary store into an empty machine. No *login* [3] or *process* persists across a boot. **boot block** the first block of a *file system* [1], which is reserved for a booting program.

break 1. an out-of-band signal on an asynchronous data line arising from the 'break' or 'interrupt' key on a terminal; before *logging in* a break causes a change in baud rate; thereafter it is interpreted as an *interrupt*. 2. a control statement in the C language. 3. the *program break*. 4. in *troff*(1), a point in running text where a new line must begin.

BSD see *UNIX*.

bss segment see *segment*.

buffer 1. a staging area for input-output where arbitrary-length transactions are collected into convenient units for system operations; the *file system* [3] uses buffers, as does *stdio*. 2. to use buffers.

buffer pool a region of store available to the *file system* [3] for holding *blocks;* all but *raw* [2] input-output for *block devices* goes through the buffer pool so read and write operations may be independent of device blocks.

cbreak a mode of terminal input in which every character not a *special character* becomes available to a *read*(2) operation as soon as it is typed, instead of being *buffered* up to a *newline* or *EOT character*.

character 1. a unit of store, usually 8 bits; a byte. 2. a token of the ASCII code, with octal value between 0 and 0177.

character device a *device* upon which a *file system* [1] cannot be *mounted*, such as a terminal or the *null device*.

child process see *fork*.

close to make an *open file* unavailable for input or output; converse of *open*.

command 1. an instruction to the *shell*, usually to run a *program* [1] as a *child process*. 2. by extension, any *executable file*, especially a *utility program*.

command file same as *shell script*.

control character an ASCII character with octal code 0-037 or 0177, which does not print but may otherwise affect the behavior of a terminal; cf. *special character*.

control terminal the terminal associated with a *process* from which the process may receive *interrupt*, *quit*, and *hangup* signals; cf. *process group*.

cooked not *raw* [1], said of an input *stream* [2] in which *special characters* are active.

cookie a peculiar goodie, token, saying, or remembrance returned by or presented to a *program* [3].

core file a *core image* of a terminated *process* saved for debugging; a core file is created under the name 'core' in the *current directory* of the process.

core image a copy of all the *segments* of a running or terminated program; the copy may exist in main store, in the *swap area*, or in a *core file*.

create to *open* a file for writing, bringing it into existence as a *plain file* if necessary, and discarding any data it may have contained previously; cf. *unlink*.

current directory, working directory the directory from which *relative pathnames* begin; a current directory is associated with each *process*.

daemon a *background* process, often perpetual, that performs a system-wide public function, e.g. *calendar*(1) and *cron*(8); the affected spelling is an ancient legacy.

Datakit (tm) a virtual circuit switch for digital

UNIX® TIME-SHARING SYSTEM

PROGRAMMER'S MANUAL

Research Version

Ninth Edition, Volume 1

September, 1986

AT&T Bell Laboratories
Murray Hill, New Jersey

PREFACE TO THE EIGHTH EDITION

This Eighth Edition manual describes the lineal descendant of the original operating system pioneered by Ken Thompson and Dennis Ritchie in the Computing Science Research Center at AT&T Bell Laboratories. It is an experimental system, not a commercial prototype, which incorporates facilities from the Seventh Edition, System V, and Berkeley BSD 4.1. The distinctive theme of the Eighth Edition is distributed computing: Ritchie's coroutine-based stream IO system and the Datakit™ virtual circuit switch realization by Lee McMahon and Bill Marshall provide the basis for networking. Peter Weinberger's remote file systems make it painless, and Rob Pike's software for the Teletype 5620 moves system action right out to the terminal. These facilities have spurred a host of programs for remote information services, interactive graphics, debugging, and simple fun. Many other people have contributed significantly, including, but not limited to L. Cardelli, T. A. Cargill, L. L. Cherry, R. J. Elliot, S. I. Feldman, F. T. Grampp, A. G. Hume, B. W. Kernighan, T. Killian, A. R. Koenig, M. E. Lesk, J. P. Linderman, B. N. Locanthi, M. S. Manasse, R. T. Morris, T. Pavlidis, D. L. Presotto, J. D. Reiser, M. J. Shannon, B. Stroustrup, and N. Wilson.

M. D. McIlroy
February, 1985

PREFACE TO THE NINTH EDITION

This revision records evolutionary changes to the landmark Eighth Edition. Like the countless corrections to the documentation, much of the substantive work embodied here is subliminal: paring away of superannuated features, simplifying the underlying model and system code, and generally improving maintainability and understandability. The touch of Norman Wilson appears everywhere. Particular attention has been given to the robustness and flexibility of security arrangements, software maintenance and distribution, internetwork communication, network management, special servers, and library rationalization by F. T. Grampp, A. G. Hume, A. R. Koenig, W. T. Marshall, D. L. Presotto, D. M. Ritchie, and K. Thompson. Graphic facilities and special databases are on the rise, as are alternative approaches to text processing. In addition to those named previously, the following people contributed significantly to the current software: J. L. Bentley, M. A. Derr, T. D. S. Duff, E. H. Gronn, G. J. Holzmann, T. J. Kowalski, S. L. Murrel, H. W. Trickey, and others. The revised manual owes much to Lorinda Cherry and Andrew Hume.

M. D. McIlroy
September, 1986

INDEX BY TOPIC

Commands and Data Files

NAME

intro – introduction to 5620-related software

DESCRIPTION

Section 9 of this manual lists software for running or supporting Teletype DMD-5620 terminals, the current implementation of the 'jerq' graphics terminals. Subsections 9.1-9.7 mirror the purposes of the preceding sections 1-7, with 9.1 being commands, 9.6 being games, etc.

Few commands deal with a 5620 in native mode. *32ld*(9.1) loads programs into the terminal and *mux*(9.1) starts the characteristic 'layer' or window system. Almost all other commands in section 9 either run on Unix or within *mux* layers.

A layer is technically a virtual terminal, but is almost indistinguishable in software from a real terminal; in particular, the interface described in *ttyld*(4) applies to layers, except for the additional editing capabilities discussed in *mux*(9.1).

The commands in sections 9.1 and 9.6 run on Unix, but most implicitly call *32ld* to down-load a program that replaces the default terminal process running in the layer. To Unix the interface is still that of a terminal; in particular */dev/tty* is always connected to the layer. The default *mux* terminal program implements the teletype function itself. When a program is down-loaded a teletype line discipline is pushed on the stream (see *stream*(4) and *ttyld*(4)). Some commands may simply emulate other terminals by down-loading a terminal program (see *term*(9.1); others, such as the text editor *sam*(9.1), are really two programs – one on Unix and one in the layer – communicating using standard input/output on Unix and `sendchar()`/`rcvchar()` in the terminal; see *request*(9.2).

There is an identity between bitmaps and layers in the graphics software. Graphic objects are bitmaps. The primitives that operate on layers are aliased to the bitmap primitives, and the data structures are isomorphic. When running under *mux*, a programmer need not consider layers as graphical objects at all; the operating system checks the arguments to the graphics primitives and dispatches the appropriate operator depending on the type of the argument. Except in stand-alone software, layers are an invisible implementation detail.

FILES

`/usr/jerq/bin`	jerq-related Unix object programs
`/usr/jerq/mbin`	terminal programs, usually down-loaded automatically by programs in `/usr/jerq/bin`

SEE ALSO

32ld(9.1), *mux*(9.1), *stream*(4), *pt*(4)

Combined Tables of Contents

The following table lists every manual page ever printed, with the editions it appeared in marked +. A number instead of a + mark indicates that the page appeared in a different chapter of that edition. An appended name in brackets [ ] means that a manual page was later superseded or subsumed by the named page.

Research software that was not included in distribution tapes was generally omitted from the v6 and v7 manuals. With v7 an addendum about unexported software was printed for local use; items from it are flagged L.

Thus one can infer from the table that *apl* existed from v5 through v7, but was never distributed. In v5 it lived in chapter 6, "User maintained commands." It disappeared with v8, a casualty of the conversion from PDP-11s to VAXes.

Trivial name changes are quietly ignored, e.g. a change from *cons*(4) in v8 to *console*(4) in v9 and from *file system*(V) in v1-v3 to *fs*(V) in v4-6 to *filsys*(5) in v7-v9. The short descriptions also changed from time to time; those given here are from v7 or else from the edition where the page first appeared.

1. Commands

Except for games, "user maintained commands" from chapter 6 of v1-v6 are included here with chapter 1, where they resided in later editions.

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . . . +++	intro	introduction to commands
. ++	:	place label [goto]
. . . . . ++	-	redo previous shell command
. +8	acct	get connect-time accounting
. . . . . + . +	adb	debugger
. . . . . + .	altran	language for algebraic computation [langs]
. . . . 6 . L . .	apl	APL interpreter
. . . . . ++	apply	apply a command to a set of arguments
. . . . . ++	apsend	send troff output to aps-5
+++++++++	ar	archive and library maintainer
. . . . . + . .	arcv	convert archives to new format
+++++++++	as	assembler
. . . . . + .	asa	interpret ASA control characters
. . . . . ++	ascii	interpret ASCII characters
. . . . . +++	at	execute commands at a later time
. . . . . +++	awk	pattern scanning and processing language
. . . 6 6 6 . . .	azel	obtain satellite predictions
+++	b	compile b program
. . . . . +	backup	backup and recover files
++++6 ++ . .	bas	basic [hoc]
. . . . . +++	basename	strip filename affixes
6 6 . . . . . + .	basic	DEC supplied basic [langs]
. . . . . ++++	bc	arbitrary-precision arithmetic language
. . . . . L . .	bs	a compiler/interpreter for modest-sized programs
. . . . . ++	bundle	collect files for distribution

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . . . + .	byteyears	time-space product for file residency
6 6 . 6 6 6 + + 7	cal	print calendar
. . . . . + + +	calendar	reminder service
. . . . . L . .	call	ring a telephone
. . . . . + +	can	interface to Cannon laser-printer spooler
+ + + + + + + + +	cat	catenate and print
. . . + 6	catsim	phototypesetter simulator
. . . . . + + +	cb	C program beautifier
. . . . . + +	cbt	btree utilities
. + + + + + + + +	cc	C compiler
. . . . . + . .	cd	change working directory [sh]
. . + + + + . . .	cdb	C debugger [adb]
. . . . . L + .	cflow	generate C flow graph
+ + + + + + . . .	chdir	change working directory [cd]
+ + + + + + + + +	chmod	change mode
+ + + + + 8 + 8 8	chown	change owner or group
. . . . . +	cin	C interpreter
. . . . . +	cite	process citations in a document
. . . . . + .	clear	clear terminal screen
+ + + + + + + + +	cmp	compare two files
. . . . 6 6 + . .	col	filter reverse line feeds [column]
. . . . . + +	column	column alignment
. . . . + + + + +	comm	select or reject lines common to two sorted files
. . . . . + . .	con	connect to another UNIX [dcon]
. . . . . + +	coreid	identify source of a core image
+ + + + + + + + +	cp	copy file
. . . . . L + +	cpio	copy file archives in and out
. 6 + + + + L . .	cref	cross-reference table
. . + . . . + + 6	crypt	encode/decode
. . . . . + +	ct	call terminal (and start a session)
. . . . . + + +	cu	call Unix
. . . . . + +	cut	rearrange columns of text
. . . . . + +	cyntax	C syntax checker
. . . . . + +	d202	phototypesetter filters
6 6	das	disassembler [adb]
+ + + + + + + + +	date	print and set the date
+ + + + + + . . .	db	symbolic debugger [adb]
+	dbppt	write binary paper tape [dump]
+ + + + + + + + +	dc	desk calculator
. . . . . + +	dcon	remote login and execution
. . . . + + + + +	dd	convert and copy a file
. . . . . + + +	deroff	remove nroff, troff, tbl and eqn constructs
+ + + 8 8 8 + + +	df	disk free
. . . . + + + + +	diff	differential file comparator
. . . . . + + .	diff3	3-way differential file comparison
. . . . . + +	dired	directory editor
6 6 8	dli	load DEC binary paper tapes
. . . . . + +	docgen	generate a document from a script
. . . . . + +	doctype	guess command line for formatting a document
6 6	dpt	read DEC ASCII paper tapes
. . . . . L . .	draw	edit a circuit diagram
. + ...	ds	verify directory hierarchy

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
+ + + + + + . . .	dsw	delete files interactively [rm]
+	dtf	format DECtape
+ + + + + + + + +	du	summarize disk usage
. + + + + + + + +	echo	echo arguments
+ + + + + + + + +	ed	text editor
. . . . . + .	efl	extended Fortran language preprocessor
. . . . + + + + +	eqn	typeset mathematics
. + + + + + . . .	exit	end command sequence [sh]
. . . . . + + +	expr	evaluate arguments as an expression
. . . . . + + +	f77	Fortran 77 compiler
. . + + 6 6 + + +	factor	factor a number, generate large primes
. + + + + + . . .	fc	compile Fortran program [f77]
. + + + + 6 . . .	fed	form-letter editor [form]
. . . . . L . .	fget	retrieve files from HIS 6000
. . . + . + + + +	file	determine file type
+ + ...	find	find file with a given name
. . . . + + + + +	find	find files
. . . . . +	fmt	ultra-simple text formatter
+	for	compile fortran program [fc]
+ + + + + 6 L . .	form	generate form letter
. . + ...	forml	generate form letters
. . . . . L . .	fsend	send files to HIS 6000
. . . . . L . .	gcat	send phototypesetter output to HIS 6000 [apsend]
. . . . . + +	getuid	get user identity
. . . . . L . .	gex	graphics exerciser for Tektronix 4014
. . . . . +	gone.fishing	automatic reply to mail
. + + + + + . . .	goto	command transfer [sh]
. . . . 6	graf	draw graph on GSI terminal
. . . . . + +	grap	pic preprocessor for drawing graphs
. . . . . 6 + + +	graph	draw a graph
. . . . . L . .	greek	interpret extended character set
. . . + + + + + +	grep	search a file for a pattern
. . . . 6 6 . . .	gsi	interpret funny characters on GSI terminal
. . . . . + +	hang	start a process in stopped state
. . . . . + +	hoc	interactive floating point language
. . . . . L . .	huff	Huffman code file compression [pack]
+	hup	hang up typewrite
. . + 6 6	hyphen	find hyphenated words
. . . . 6 . L . .	ibm	submit off-line job to HO IBM 370
. . . . . + .	icont	Icon language translator and compiler
. . . . . + +	ideal	troff preprocessor for drawing pictures
. . . . . + +	idiff	interactive file comparison
. + + + + + . . .	if	conditional command [sh]
. . . . . L . .	iget	get files from Holmdel IBM 370
. . . . . + + .	iostat	report I/O statistics [load]
. . . . . L . .	isend	send files to Holmdel IBM 370
. . . . . + + +	join	relational database operator
. . 8 + + + + + +	kill	terminate a process with extreme prejudice
. . . . . L + +	lab	label maker
. . . . . +	langs	altran, basic, ... languages
. . . . . + .	last	report recent logins [who]
+	lbppt	read binary paper tape [restor]

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . . . ++	lcomp	line-by-line profiler
+++++++++	ld	loader
. . . . . L . .	lde	logic design equation language
. . . . . +++	learn	computer aided instruction about UNIX
. . . . . +++	lex	generator of lexical analysis programs
. . . . . +++	lint	a C program verifier
. . . . . +.	lisp	lisp interpreter and compiler [langs]
+++++++++	ln	make a link
. . . . . +	load	load and input-output statistics
. ++++++8 8	login	sign on
. . . . . +++	look	find lines in a sorted list
. . . . . +. .	lookall	look through all text files on UNIX
. . . . . +. +	lorder	find ordering relation for an object library
. . . . +. +++	lpr	line printer spooler
+++++++++	ls	list contents of directory
. . . . . +++	m4	macro processor
. ++666 . . .	m6	macroprocessor [m4]
. . . . . ++	Mail	send and receive mail
+++++++++	mail	send or receive mail among users
. . . . . +++	make	maintain program groups
. +++++++++	man	print sections of this manual
. . . . . +.	matlab	interactive matrix desk calculator [langs]
. . . +.	merge	merge several files [sort]
+++++++++	mesg	permit or deny messages
. . . . . +	mk	maintain (make) related files
+++++++++	mkdir	make a directory
. . . . . +	mkpkg	make and install packages
. . . . . +	monk	typeset documents and letters
. ++.	mt	save/restore files on magtape [tar]
+++++++++	mv	move or rename files and directories
. . . . ++. . .	neqn	typeset mathematics on a terminal [eqn]
. . . . . ++	newer	test file modification dates
. . . . . ++++	newgrp	log in to a new group
. . . . . ++	news	print news items
. . . . . L . .	nfs	communicate with Spider File System
. . . ++++++	nice	run a command at low priority
+++++++++	nm	print name list
. . . +++. . .	nohup	run a command at low priority [nice]
. . . . 6	npr	print file on Spider line-printer
. +++++. . .	nroff	format text for printing [troff]
. . . . . L+6	number	convert Arabic numerals to English
+++++++++	od	octal dump
. +++++L . .	opr	print file off-line
. ++6	ov	page overlay file print
. . . . . ++	p	paginate
. . . . . ++	pack	compress and expand files
. . . . . +.	paper	list input on HP2621P printer
. . . . . ++	pascal	language interpreter
. . +++++++	passwd	install new password or user
. . . . . +	pc	pascal language compiler
. . . +++: . .	pfe	print floating exception
. . . . . ++	pic	troff preprocessor for drawing pictures

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . . . + .	pick	pick arguments [apply]
. . . . . L . .	place	design physical layout of a circuit
. . . . 6	plog	make a graph on the gsi terminal
. . . + 6 6 + + +	plot	graphics filter
. . . . . + .	post	send mail to users by name
+ + + + + + + + +	pr	print file
. . . . . +	prefer	maintain and use bibliographic references
. . . . . + . .	prep	prepare text for statistical processing
. . . . . 6 . . .	primes	print all primes larger than somewhat [factor]
. . . . + + + + +	prof	display profile data
. . . . . L . .	prom	read and write proms through the PROLOG promwriter
. . + + ...	proof	compare text files [diff]
. . 8 + + + + + +	ps	process status
. 6 6 6 6 . + + +	ptx	permuted index
. . . . . + . .	pubindex	make inverted bibliographic index [refer]
. . . . . + +	push	datakit remote file copy
. . . . + + + + +	pwd	working directory name
. . . . . + .	pxp	pascal printer, profiler, and cross-reference lister
. . . . . + +	random	sample lines from a file or provide random exit code
. . . . . + .	ranlib	convert archives to random libraries [ar]
. . . . . + + + .	ratfor	Ratfor compiler [langs]
. . . . . + + +	refer	find and insert literature references in documents
. . . . . + + + +	rev	reverse lines of a file
+ + + + + ...	rew	rewind DECtape
+	rkd	dump disk to tape
+	rkf	format RK disk
+	rkl	load disk from tape
+ + + + + + + + +	rm	remove (unlink) files
+ + + + + + . . .	rmdir	remove (delete) directory [rm]
+ + + + + + + . .	roff	format text
. . . . . + +	ropy	remote file copy for arpa internet
. . . . . +	rscan	scan pages on ricoh scanner and display on 5620
+	sdate	adjust date and time [date]
. . . . . + +	sdb	symbolic debugger
. . . . . L . .	sdiff	side-by-side difference program
. . . . . +	seal	mailable data file
. . . . . + + +	sed	stream editor
. . . . . +	sendcover	send cover sheet to the library
. . . . . + +	seq	print sequences of numbers
. . . . . + +	server	run anonymous command on another machine
. . . . 6	sfs	structured file scanner
+ + + + + + + + +	sh	command language
. . + + + + . . .	shift	adjust shell arguments [sh]
. . . . . + +	ship	automatic software distribution
. . + + + + + + +	size	size of an object file
. . . 6 6 6 L 7 7	sky	obtain ephemerides
. . . + + + + + +	sleep	suspend execution for an interval
. . + + 6 6 L + .	sno	compile Snobol programs [langs]
. . . . . + +	snocone	snobol with syntactic sugar
6 + + + + + + + +	sort	sort or merge files
. . + + 6 6 L . .	speak	send words to voice synthesizer
. . . . + + + + +	spell	find spelling errors

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . . . + .	spitbol	Snobol language compiler [langs]
. . . 6 6 6 + . .	spline	interpolate smooth curve
. . + + + + + + +	split	split a file into pieces
+ + + ...	stat	get file status
+ + + + + + + + +	strip	remove symbols and relocation bits
. . . . . + + .	struct	structure Fortran programs
. + + + + + + + +	stty	set terminal options
+ + + + + . + + +	sum	sum and count blocks in a file
. . . . . + + +	tabs	set terminal tabs
. . . . . + + +	tail	deliver the last part of a file
+ + + ...	tap	manipulate DECtape
. . . . . + +	tape	identify and manipulate magnetic tape
. . . . . + + +	tar	tape archiver
. . . . . 6 + + +	tbl	format tables for nroff or troff
. . . . . + . .	tc	troff output interpreter
. . . . + + + + +	tee	pipe fitting
. . . . . L . .	tekstare	convert tektronix picture to hard copy graphics [can]
. . . . . + .	telnet	user interface to the telnet protocol
. . . . . + + +	test	condition command
. . . . . +	tex	text formatting and typesetting
. . + + + + + + +	time	time a command
. . . . . + + .	tk	paginator for the Tektronix 4014
. 6 + 6 6 6 L . .	tmg	compile tmgl program
. . . . . + + +	touch	update date last modified of a file
. . . + + + + . .	tp	manipulate tape archive [tar]
. . . + + + + + +	tr	translate characters
. . . . . + +	trace	protocol compiler and analyzer
. . . . . + .	track	selective remote file copy
. . . + + + + + +	troff	text formatting and typesetting
. . . . . + + +	true	provide truth values
. . . . . + .	tset	set terminal modes
. . . . . + + +	tsort	topological sort
. + + + + . L . .	tss	communicate with MH-TSS (GCOS)
+ + + + + + + + +	tty	get terminal name
+ + + + ...	type	print file on IBM 2741
. . + + + + L . .	typo	find typographic errors
. . . . . L . .	ufs	Spider Network Communication
. . . . . + +	ul	print underlines on screen terminals
+ + + ...	un	fine undefined symbols
. . + + + + + + +	uniq	report repeated lines in a file
. . . . . 6 + 7 7	units	conversion program
. . . . . + . . .	usort	sort and merge files, discarding duplicate lines [sort]
. . . . . + + +	uucp	unix to unix copy
. . . . . L . .	uudiff	directory comparison between machines
. . . . . + +	uustat	uucp status inquiry and job control
. . . . . + + +	uux	unix to unix command execution
. . . . . L . .	vc	verification of tests for C programs [lcomp]
. . . . . + +	vi	screen oriented (visual) display editor based on ex
. . . . . + +	view2d	movie of a function f(x,y,t)
. . . . . + +	vis	show invisible characters
. . . . . + .	visi	mathematical spreadsheet
. . + ...	vs	generate voice synthesizer phonemes

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . + + + + . .	wait	await completion of process [sh]
+ + + + + + + + +	wc	word count
. . . . . L . .	wcheck	look for inconsistencies in a circuit description
+ + + + + + + + +	who	who is on the system
. . . . . L . .	wrap	generate control information for wiring a circuit board
+ + + + + + + + +	write	write to another user
. . . . . + +	wwb	writers workbench
. . . . . + +	wwv	print and set the date from accurate clock
. . . . . L . .	xref	cross reference for C programs
. . . . . + + .	xsend	secret mail
. . 6 6 6 + + + +	yacc	yet another compiler-compiler

2. System calls

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . . + + + + +	intro	introduction to system calls and error numbers
. . . . . + + +	access	determine accessibility of file
. . . . . + + +	acct	turn accounting on or off
. . . . . + + +	alarm	schedule signal after specified time
. . + ... + .	boot	reboot the system
+ + + + + + . . .	break	set program break [brk]
+ + + ...	cemt	catch EMT traps [signal]
+ + + + + + + + +	chdir	change default directory
+ + + + + + + + +	chmod	change mode of file
+ + + + + + + + +	chown	change owner and group of a file
+ + + + + + + + +	close	close a file
+ + + + + + + + +	creat	create a new file
. . + + + + . . .	csw	read the console switches
. . . . . +	deprecated	system calls to be avoided
. . + + + + + + +	dup	duplicate an open file descriptor
+ + + + + + + + +	exec	execute a file
+ + + + + + + + +	exit	terminate process
. . . . . +	fmount	mount or remove file system
+ + + + + + + + +	fork	spawn new process
. . + ...	fpe	catch floating exception errors [signal]
+ + + + + + . . .	fstat	status of open file [stat]
. . . + + + . . .	getgid	get group identification [getuid]
. . . . . + + . .	getpid	get process identification [getuid]
+ + + + + + + + +	getuid	get user and group identity
. . . . . + .	gmount	mount or remove non-standard file system [fmount]
+ + + + + + . . .	gtty	get typewrite mode [ioctl]
. + ...	hog	set low-priority status [nice]
+ + + ...	ilgins	catch illegal instruction trap [signal]
. . . + + + + . .	indir	indirect system call [syscall]
+ + + ...	intr	catch or inhib interrupts [signal]
. . . . . + + +	ioctl	control device
. + + + + + + + +	kill	send signal to a process
+ + + + + + + + +	link	link to a file
. . . . . + . .	lock	lock a process in primary memory
. . . . . + + +	lseek	move read/write pointer
+ + + ...	mkdir	create directory
. . . + + + + + +	mknod	make a directory or a special file

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
+ + + + + + + + .	mount	mount or remove file system [fmount]
. . . . . + . .	mpx	create and manipulate multiplexed files
. . + + + + + + +	nice	set program priority
+ + + + + + + + +	open	open for reading or writing
. . . . . + + .	pause	stop until signal [alarm]
. . . . . + . .	phys	allow a process to access physical addresses
. . + + + + + + .	pipe	create an interprocess channel
. . . . . + . .	pkon	establish packet protocol
. . . . + + + + +	profil	execution time profile
. . . . . + + + .	ptrace	process trace [proc(4)]
+ + + ...	quit	catch or inhibit quits [signal]
+ + + + + + + + +	read	read from file
+ + + ...	rele	release processor
+ + + + + + . . .	seek	move read or write pointer [lseek]
. . . . . + +	select	synchronous I/O multiplexing
. . . + + + . . .	setgid	set process group ID [setuid]
+ + + + + + + + +	setuid	set user and group ID
. . . + + + + + +	signal	catch or ignore signals
. + + + + + . . .	sleep	delay execution [alarm]
+ + + ...	smdate	set date modified of file [utime]
+ + + + + + + + +	stat	get file status
+ + + + + + + + +	stime	set time
+ + + + + + . . .	stty	set mode of typewriter [ioctl]
. + + + + + + + +	sync	update super-block
. . . . . + +	syscall	indirect system call
+ + ...	tell	find read or write pointer [seek]
+ + + + + + + + +	time	get date and time
. . + + + + + + +	times	get process times
. . . . . + +	umask	set file creation mode mask
+ + + + + + . . .	umount	dismount file system [mount]
+ + + + + + + + +	unlink	remove directory entry
. . . . . + + +	utime	set file times
+ + + + + + + + +	wait	wait for process to terminate
+ + + + + + + + +	write	write on a file

3. Subroutines

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . . . + + +	intro	introduction to library functions
. . . . . + + + +	abort	generate IOT fault
. . . . . + + . .	abs	integer absolute value, sign function [arith]
. . . . . + + . . .	alloc	core allocator [malloc]
. . . . . + +	arith	integer arithmetic functions
. . . . . + + +	assert	program verification
. + + + + + . . .	atan	arctangent [sin]
+ + + + + + + + +	atof	convert ASCII to numbers
+ + + . . + . . .	atoi	convert ASCII to integer [atof]
. . . . . + +	cbt	compressed B-tree subroutines
. . . . . + +	chrtab	simple character bitmaps
. . + + ...	compar	string compare for sort
. + ...	const	floating point constants
. . . . . 7 L . .	cr	coroutine scheme

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . + + + + + + +	crypt	DES encryption
+ + + + + + + + +	ctime	convert date and time to ASCII
. . . . . + + +	ctype	character classification
. . . . . + +	curses	screen functions with 'optimal' cursor motion
. . . . . + + +	dbm	data base subroutines
. . + ...	ddsput	display characters on Picturephone
. . . . . + +	dialout	place call on ACU
. . . . . + +	directory	directory operations
. . . . . + .	dkmgr	establish datakit server
. . + + + + + + +	ecvt	output conversion
. . . . . + + + +	end	last locations in program
. . . . . L + +	erf	error function
+ + + + + + + + +	exp	exponential, logarithm, power, square root
. . . . . + + +	fclose	close or flush a stream
. . . . . + + +	ferror	stream status inquiries
. . . . . +	fio	fast buffered I/O
. . . . + + + + +	floor	absolute value, floor, ceiling functions
. . . . . + . . .	fmod	floating modulus function [floor]
. . . . . + + +	fopen	open a stream
+ + + + + + . . .	fptrap	floating-point simulator
. . . . . + + +	fread	buffered binary input/output
. . . . . + + +	frexp	split into mantissa and exponent
. . . . . + + +	fseek	reposition a stream
+ + + ...	ftoa	convert floating to ASCII [ecvt]
. . + ...	ftoo	convert floating to octal
. . . . . + +	ftw	file tree walk
. . . . . L + +	galloc	storage allocation with garbage collection
. . . . + + L + +	gamma	log gamma function
. + + + ...	gerts	communicate with GCOS
. . . . + + . + +	getarg	get command arguments from Fortran
+ + + + + + + + +	getc	get character or word from stream
. . . + + + . . .	getchar	read character [getc]
. . . . . + + +	getenv	value for environment name
. . . . . +	getfields	break a string into fields
. . . . . + +	getfsent	get file system descriptor file entry
. . . . . + + +	getgrent	get group file entry
. . . . . + + +	getlogin	get login name
. . . . . + +	getopt	get option letter from argv
. . . . . + + +	getpass	read a password
. . . + + + + . .	getpw	get name from UID
. . . . . + + +	getpwent	get password file entry
. . . . . + + +	gets	get a string from a stream
. . . . . + +	getwd	get current directory
. . . + + + . . .	hmul	high-order product
. + + + + . + + +	hypot	euclidean distance
. . . + + + . . .	ierror	catch Fortran errors
. . . . . +	internet	internet networking functions
. . . . . +	ipc	set up communications between unrelated processes
+ + + ...	itoa	convert integer to ASCII
. . . . . + + +	j0	bessel functions
. . . . . + + +	l3tol	convert between 3-byte integers and long integers
. . . + + + . . .	ldiv	long division

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
7	liba	standard assembly-language library
7	libb	standard B library
7	libf	standard Fortran library
. . . . + + . . .	locv	long output conversion [printf]
+ + + + + + . . .	log	logarithm base e [exp]
. . . . . + + +	malloc	main memory allocator
. . . . . L + +	map	map projections
. . . . . + +	memory	memory operations
+ + + + ...	mesg	print string on typewriter [printf]
. . . . . + + +	mktemp	make a unique file name
. . . . + + + + +	monitor	prepare execution profile
. . . . . + + +	mp	multiple precision integer arithmetic
. . . + + + . . .	nargs	argument count
. + + + + + + + +	nlist	get entries from name list
. . . + + + + + +	perror	system error messages
. . . . . + . .	pkopen	packet driver simulator
. . . . . 7 + + +	plot	graphics interface
. . . . . + + +	popen	initiate I/O to/from a process
. . . . . + +	port	mathematical library for Fortran
. . + + + + . . .	pow	take powers of numbers [exp]
. . . . . +	print	print formatted output
. . . + + + + + +	printf	output formatters
+ + + ...	ptime	print time
. . . . . + +	ptopen	find and open a pseudo-terminal file
+ + + + + + + + +	putc	put character or word on a stream
. . . + + + . . .	putchar	write character [putc]
. . . . . + + +	puts	put a string on a stream
. + + + + + + + +	qsort	quicker sort
. . + + + + + + +	rand	random number generator
. . . . . + .	regex	regular expression handler [regexp]
. . . . . + +	regexp	regular expression handler
. . . + + + . . .	reset	execute non-local goto [setjmp]
. + + . . 7 L . .	salloc	string allocation and manipulation
. . . . . + + +	scanf	formatted input conversion
. . . . . + + +	setbuf	assign buffering to a stream
. . . + + + . . .	setfil	specify Fortran file name
. . . . . + + +	setjmp	non-local goto
+ + + + + + + + +	sin	trigonometric functions
. . . . . + + +	sinh	hyperbolic functions
. . . . . + + +	sleep	suspend execution for interval
. + + + + + . . .	sqrt	square root [exp]
. . . . . + + +	stdio	standard buffered input/output package
. . . . . + + +	string	string operations
. . . . . + + +	swab	swap bytes
+ + + + ...	switch	transfer depending on value
. . . . . + + +	system	issue a shell command
. . . . . +	tcp	tcp networking functions
. . . . . L + +	tdkdial	open a datakit connection to a remote server
. . . . . + +	termcap	terminal independent operation routines
. . . . . + +	tolower	force upper or lower case
. . + + + + + + +	ttyname	find name of a terminal
. . . . . +	udp	udp networking functions

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . . . +	uname	get password file entry
. . . . . + + +	ungetc	push character back into input stream
. . . . . + +	varargs	variable argument list
. . . . . + +	view2d	movie of a function f(x,y,t)
. . . + + . . .	vt	display (vt01) interface

4. Special files

Terminology changed often in this section. In v3 mnemonic names were replaced by names of DEC devices, for example, *tty* became *kl*, *tty0* became *dc*, and *ppt* became *pc*. The more recent names are used here. Lately the trend has reversed, with the appearance of *drum* and *cons*.

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . . . + +	bufld	buffering line discipline
. . . + + + + . .	cat	phototypesetter interface
. . . . . +	connld	connection line discipline
. . . . . + +	cons	console interface
. . . + ...	da	voice response unit
+ + + + + + . . .	dc	remote typewriter
. . . . + + . . .	dh	DH-11 communications multiplexor
. . . . . L + +	dk	Datakit interface
. + + + + + + . .	dn	DN-11 ACU interface
. + + + + + . . .	dp	201 dataphone
. . . . . + +	drum	paging device
. . . . . + . .	du	DU-11 201 data-phone interface
. . . . . + +	fd	file descriptor file
. . . . . + + . .	hp	RH-11/RP04, RP05, RP06 moving-head disk
. . . . . + + . .	hs	RH11/RS03-RS04 fixed-head disk file
. . . . . + + . .	ht	RH-11/TU-16 magtape interface
+ + + + + + . . .	kl	console typewriter [cons]
. + . . + + L + .	lp	line printer
+ + + + + + + + .	mem	core memory
. . . . . + +	mesgld	message line discipline
. + + + + + + + +	mt	magtape interface
. . . . . + . + +	null	data sink [mem]
+ + + + + + . . .	pc	punched paper tape
. . . . . + . .	pk	packet driver
. . . . . + +	proc	process file system
. . . . . + +	pt	interprocess I/O junctor files
. . . . . + +	ra	DEC MSCP disks (RA60, RA80, RA81)
+ + + + + + + . .	rf	RF11/RS11 fixed-head disk file
+ + + + + + + + .	rk	RK-11/RK03 or RK05 disk
. + . + + + + . .	rp	RP-11/RP03 moving-head disk
. . . . . + +	stream	stream I/O control calls
+ + + + + + + . .	tc	TC-11/TU56 DECtape
. . . + + . L . .	tiu	Spider interface
. . . . + + + . +	tty	general terminal interface [ttyld]
. . . . . + +	ttyld	terminal processing
. . . . . + . .	vp	Versatec printer-plotter
. . . + + . L . .	vs	voice synthesizer interface

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . +++. . . .	vt	storage-tube display

5. File formats and conventions

In v1-v5, section 5 was restricted to "File formats"

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
+++++++++	a.out	assembler and link editor output
. . . . . +++	acct	execution accounting file
+++++++++	ar	archive (library) file format
. . . . . +	backup	incremental backup file
+.	bppt	binary paper tape format
. . . . . +	config	system configuration files
+++++++++	core	format of core image file
. . . . . +	cpio	format of cpio archive
+++++++++	dir	format of directories
. . . . +++. .	dump	incremental dump format
. . . . . +++	environ	user environment
+++++++++	filsys	format of file system volume
. . . . . ++	fstab	static information about the file system
. . . . . ++++	group	group file
. +.	ident	GCOD ident cards
. . . . . ++	map	digitized map formats
. . . . . +. .	mpxio	multiplexed i/o
. . . . +++++	mtab	mounted file system table
. . . . . ++	netnews	USENET network news article, utility files
. . . . . +.	newsrc	information file for readnews
+++++++++	passwd	password file
. . . . . 7 +++	plot	graphics interface
. . . . . +	poly	polyhedra database format
. . . . +. . . .	speak.m	voice synthesizer vocabulary
. . . . . ++	stab	symbol table types
. . . . . L . .	tar	format of tar archive
. . . . . ++	termcap	terminal capability database
. ++++++. .	tp	DEC/mag tape formats
. . . . +++++	ttys	terminal initialization data
. . . . . ++	types	primitive system types
++.	uids	map names to user ID's
+++++++++	utmp	login records
. . . . . ++	view2d	movie of a function f(x,y,t)
. . . . . ++	whoami	computer name
. +++++. . .	wtmp	accounting files [utmp]

6. Games

In v1 through v6 chapter 6 was called "User maintained maintained programs." Only the games from those editions are listed here; other pages from those chapters 6 are listed with chapter 1 or chapter 7.

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . . . L ++	adventure	dungeon-exploration game
. . . . . +++	arithmetic	provide drill in number facts
. . . . . ++	atc	air traffic controller

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . . . + + +	backgammon	the game
. . . . . + + +	banner	make long posters
1 . . . . . + + +	bcd	convert to antique media
+ + + + + + + . .	bj	the game of black jack
. . . . . + +	boggle	word games
. . . . . + +	bridge	card game
. . . . . + +	cards	card games
. . . . . + . .	checkers	game
+ . . + + + + . .	chess	the game of chess
. . . . . + + .	ching	the book of changes and other cookies
. . . + + + . . .	cubic	three dimensional tic-tac-toe
. . . . . + +	doctor	psychiatric consultation
. . . . . +	festoon	memo writer
. . . . . + +	fortune	cookies
. . . . . + . . .	king	the game of king
. . . + + . + . .	maze	generate a maze problem
+ + . + + + + . .	moo	guessing game
. . . . . L . .	morse	convert letters to morse code
. . . . . 1 +	netnews	sen dor receive news article
. . . . . L . .	psych	pattern generators
. . . . . + + + +	quiz	test your knowledge
. . . . . + . .	reversi	a game of dramatic reversals
. . . . . + +	snake	display chase game
. . . . . L + +	trek	war games
+ + . + + + + . .	ttt	tic-tac-toe
. . . . . + . .	words	word games [boggle]
. . . . . + +	worms	silly demos
. . . . + + + . .	wump	the game of hunt-the-wumpus

7. Data bases and language conventions

Like chapter 6, chapter 7 has been a catch-all, with various names over the years:

v1-v5 Miscellaneous
v6 User maintained subroutines
v7 Conventions
v8-9 Data bases and language conventions

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . . . + +	apnews	present ap wire stories
+ + + + + 5 + + +	ascii	map of ASCII character set
. . . . . + .	candest	canon laser printers [can(1)]
. . . . . L . .	cdl	circuit description language
. . . . . + +	dict	look up words in English dictionaries
. . . . . + + +	eqnchar	special character definitions for eqn
. . . . . + +	font	typesetter fonts
. . + + + 5 + . .	greek	graphics for extended HdY-37 type-box
. . . . . + . .	hier	file system hierarchy
+ + ...	kbd	map of HdY 37 keyboard
. . . . . +	latex	tex macro packages and bibliographies
. . . . . + +	library	bell labs library service
+ + ...	login	logging on and logging off the system [how to get started]
. . . . . + +	mail	address conventions and rewrite rules

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . . . +++	man	macros to typeset manual
. . . . . L ++	map	draw maps on various projections
. . . . . +	mbits	macros for typesetting bitmaps
. . . . . ++	mcs	macros for formatting cover sheets
. . . . . ++++	ms	macros for formatting manuscripts
. . . . . +	netnews	recent articles, utility files
. . . . . +	papers	browse database of locally authored papers
. . . . . +	poly	database of polyhedra
. . . . . 1 +	postnews	submit netnews articles
. . . . . 1 +	readnews	read news articles
. . . . . +	scat	sky catalog
. . . . . +	submit	install document in database
+	suftab	roff's suffix table
+++++5 . . .	tabs	set tab stops on typewriter [tabs(1)]
. . . . . ++	tel	local and private telephone books
. . . . . + .	telno	retrieve from bell labs phone book [tel]
. . . . . ++ .	term	conventional names
. . . ++	tmheader	TM cover sheet
. . . . . ++	town	gazetteer of US places
. . . . . ++	troff	addenda to troff manual
. . +++ . L . .	vsp	voice synthesizer phonemes
. . . . . ++	weather	conditions and forecast by town

8. Maintenance commands and procedures

Pages from chapter 1 of v1, v2, and v7 that appeared in chapter 8 of other editions are included here. In v1 and v2 there was no chapter 8 and in v7 many system maintenance commands were placed in chapter 1, with the identification "1M".

Chapter 8 is the most turbulent part of the manual: maintenance procedures, being known only to a few, and often being embedded in just one or two shell scripts may be more lightly changed than mainstream facilities. Moreover, much of chapter 8 is concerned with hidden procedures that are usually invoked automatically.

It has alway been problematic just how much to say about such changeable things that so few people need to know about. Maintenance programs may remain "unofficial" for years. For example, one or another version of *findo*, for scouring trash out of full file systems, had existed since the earliest days, yet it was not documented until v8.

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . . . ++	11	pdp11 support
1 7 +++	20boot	rebooot 11/20 system
. . . . ++ 1 ++	ac	login accounting
. . . . . +	arff	read RT11 files
7	as2	assembler's pass 2
. . . . . ++	asd	automatic software distribution
7	ba	B assembler
. . . . . +	backup	backup administration
7	bc	B compiler
7	bilib	B interpreter
7 7 +++++ . .	boot	startup procedures [reboot]
7	brt1,brt2	B start and finish
. 6	chash	prepare symbol table
1 1 +++	check	check consistency of file system [icheck]

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . . . + . . .	chgrp	change group [chown]
. . + ...	chk	check all file systems
. . 1 + + + 1 + +	clri	clear i-node
. . . . . + +	cpp	C language preprocessor
. . . . . + + . .	crash	what to do when the system crashes
. . . . . + + + +	cron	clock daemon
. . + . . + 1 . .	dcheck	file system directory consistency check [icheck]
. . . . . + +	dmesg	system diagnostic messages
. 1 7 7 + + L . .	dpd	dataphone daemons
. . . . + + 1 . .	dump	incremental file system dump
. . . . . 1 . .	dumpdir	print the names of files on a dump tape
7	f1,f2,f3,f4	Fortran compiler passes
. . . . . L . .	fget.demon	fget daemons
. . . . . + +	finddev	find process using a device
. . . . . + +	findo	find objectionable files
. . . . . + +	fsck	file system consistency check and interactive repair
. 7 7 7 + + + + +	getty	set typewriter mode
7 7 7 7 + + . . .	glob	argument expander
. . . . . + 1 + +	icheck	file system storage consistency check
7 7 7 7 + + + + +	init	process control initialization
. . . + ...	ino	get the i-number of a file
. 1 + ...	istat	file status by i-number
. . . . . +	ldpcs	load correct microcode
. . . . + + L + +	lpd	line printer daemon
. . . . . + + +	makekey	generate encryption key
. . . . . +	mgrproc	service remote computing requests
. . . . . 1 . .	mkconf	generate configuration tables [config]
1 . . + + + 1 + +	mkfs	construct a file system
. . . + + + 1 + +	mknod	build special file
1 1 + + + + 1 + +	mount	mount and dismount file system
7 7 7 7 + ...	msh	mini Shell
. . . . . + 1 . .	ncheck	generate names from i-numbers [icheck]
. . . . . + +	netfs	network file system
. . . . . + +	netstat	show network status for ARPA internet
. . . . . + +	oops	process status
. . . . . 1 + +	pstat	print system facts
. . . . . 1 + +	quot	summarize file system ownership
. . . . . + +	rarepl	replace bad blocks on MSCP drive
. . . . . + +	rc	boot script
. . . . . + +	reboot	bootstrapping procedures
. . 1 + + ...	reloc	relocate object files
. . . . . + +	renice	alter priority of running process by changing nice
. . . . + + 1 . .	restor	incremental file system restore
. . . . . + .	rmdir	unlink directory
. . . . + + 1 + +	sa	system accounting
. 1 + ...	salv	repair damaged file system
. . . . . + +	savecore	save a core dump of the operating system
. . . . . + +	showq	state of stream I/O system
. . . . . +	smash	rewrite bad disk sectors
1 1 + + + + 1 + +	su	substitute user id temporarily
. . . . . + +	swapon	specify paging/swapping device
. . . + + + 1 + +	sync	update the super block

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
1 1 + ...	tm	get time information
1 1 + + + + . . .	umount	dismount removable file system [mount]
. . . . .	+ + upas	address driven mailer
. . . + + + + + +	update	periodically update the super block
. . . . .	+ + uucheck	check uucp directories and permissions file
. . . . .	+ + uucico	file transport program for the uucp system
. L	+ + uuclean	uucp spool directory cleanup
. . . . .	+ . uusched	uucp file transport scheduler [uucico]
. . . . .	+ + uuxqt	create remote command requests
. . . . .	+ + vmstat	report virtual memory statistics
. . . . . + 1	+ + wall	write to all users
. . . . .	+ . xstr	preprocessor for sharing strings in C programs

9. Teletype 5620-related software

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . . .	+ + intro	introduction to jerq-related software
9.1 Commands		
. . . . .	+ + 32ld	bootstrap loader for the 5620
. . . . .	+ + 3cc	MAC-32 compiler for the 5620
. . . . .	+ + blitblt	make hard copy image
. . . . .	+ brush	painting program
. . . . .	+ + cip	picture drawing program
. . . . .	+ + face	show faces on a jerq
. . . . .	+ flicks	movie graphics for 5620
. . . . .	+ getfont	replace terminal's default font
. . . . .	+ + graphdraw	edit (combinatoric) graph
. . . . .	+ + icon	icon editor
. . . . .	+ + jf	font editor
. . . . .	+ . jim	text editor [sam]
. . . . .	+ + jx	jerq execution and stdio interpreter
. . . . .	+ + lens	bitmap magnifier
. . . . .	+ + mugs	convert gray-scale images into icons
. . . . .	+ + mux	layer multiplexor for the jerq
. . . . .	+ + paint	draw pictures in a layer
. . . . .	+ + ped	picture editor
. . . . .	+ + pi	process inspector
. . . . .	+ + pico	graphics editor
. . . . .	+ + proof	troff output interpreter for jerq
. . . . .	+ pvmon	gray-scale picture preview window for 5620
. . . . .	+ reader	examine typeset documents
. . . . .	+ + rebecca	graphics touch-up editor
. . . . .	+ + ruler	measure things on the screen
. . . . .	+ sam	screen editor with structural regular expressions
. . . . .	+ + term	nonstandard mux terminals
. . . . .	+ + thinkblt	print on ThinkJet
. . . . .	+ + vismon	system statistics and mail notification
. . . . .	+ + windows	create and initialize windows

9.2 System calls

The shortness of this subchapter and the next belies the comprehensiveness of the *mux* operating system. Descriptions were combined to save pages: the four pages of 9.2 document some three dozen system calls.

Edition 1 2 3 4 5 6 7 8 9	Title	Purpose
. . . . . ++	button	mouse control
. . . . . ++	newlayer	layer control and graphics
. . . . . ++	newproc	jerq process control
. . . . . ++	request	jerq I/O requests
9.3 Subroutines		
. . . . . ++	add	arithmetic on points and rectangles
. . . . . ++	alloc	allocate memory
. . . . . ++	bitblt	basic jerq graphics functions
. . . . . ++	circle	circle drawing functions for jerq
. . . . . ++	cos	integer math functions
. . . . . ++	menuhit	present user with menu and get selection
. . . . . ++	string	jerq text and font operations
. . . . . +	thinkclient	ThinkJet routines
9.4 Devices		
. . . . . ++	jioctl	jerq ioctl requests
. . . . . ++	mouse	jerq mouse interface
9.5 File formats and conventions		
. . . . . ++	bitfile	format of bitmap file
. . . . . ++	faced	network face server
. . . . . ++	font	jerq font layouts
. . . . . +	movies	graphics movie file formats
. . . . . ++	pads	user interface package
. . . . . ++	types	basic jerq graphics data types
9.6 Games		
. . . . . ++	crabs	graphical marine adventure game
. . . . . ++	demo	graphic demonstration and games
. . . . . +	gebaca	get back at corporate america
. . . . . +	menudrop	leave a menu lying around
. . . . . ++	pen	doodle anywhere on the screen
. . . . . +	pengo	squash the sno-bees
. . . . . ++	twid	dabble in oils
9.7 Data bases		
. . . . . ++	blitmap	map plots and path finding on a jerq

References

1. AT&T Bell Laboratories *UNIX System Readings and Applications*. Prentice-Hall, Englewood Cliffs, NJ, 1986.

2. Bell Telephone Laboratories *UNIX Programmer's Manual*. Vol. 1. Holt, Rinehart and Winston, New York, 1983.

3. Feldman, S.I. An Architecture History of the UNIX System. In *USENIX Conference*, Salt Lake City, Summer 1984.

4. Grampp, F.T. and Morris, R.H. UNIX Operating System Security. *AT&T Bell Laboratories Technical Journal 63*, 8, Pt. 2 (1984), 1649-1672. UNIX special issue. Reprinted in [1], Vol. II, 18-28.

5. Kernighan, B.W. and Pike, R. *The UNIX Programming Environment*. Prentice-Hall, Englewood Cliffs, NJ, 1984.

6. Kernighan, B.W. and Plauger, P.J. *Software Tools*. Addison-Wesley, Reading MA, 1976.

7. McIlroy, M.D. The UNIX Success Story. *UNIX Review 4*, 10 (October 1986), 32-42.

8. McMahon, L.E., Cherry, L.L., and Morris, R. Statistical Text Processing. *Bell System Technical Journal 56*, 6, Pt. 2 (1978), 2137-2154. UNIX special issue. Reprinted in [1], Vol.I, 227-244.

9. Pike, R. and Kernighan, B.W. Program Design in the UNIX System Environment. *AT&T Bell Laboratories Technical Journal 63*, 8, Pt. 2 (1984), 1598-1601. UNIX special issue. Reprinted in [1], Vol. II, 21-24.

10. Ritchie, D.M. Reflections on Software Research. *Comm. of the ACM 27*, 8 (1984), 758-760.

11. Ritchie, D.M. The Evolution of the UNIX Time-sharing System. *AT&T Bell Laboratories Technical Journal 63*, 8, Pt. 2 (1984), 11-17. UNIX special issue. Reprinted in [1], Vol. II, 11-17.

12. Ritchie, D.M. and Thompson, K. The UNIX Time-Sharing System. *ACM Operating Systems Review 7*, 4 (October 1973), 1-9. Proceedings of Fourth Symposium on Operating System Principles, Yorktown Heights NY.

13. Ritchie, D.M. and Thompson, K. The UNIX Time-Sharing System. *Comm. of the ACM 17*, 7 (July 1974), 365-375. Also *Bell System Technical Journal 57*, 6 (1978), 1905-1929. UNIX special issue. Reprinted in [1], Vol. I, 1-25.